UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GIOVANE GONÇALVES DA SILVA

AVALIAÇÃO DO IMPACTO DO TLS NA PERFORMANCE DA COMUNICAÇÃO MQTT USANDO MOSQUITTO

CHAPECÓ
2026

GIOVANE GONÇALVES DA SILVA

AVALIAÇÃO DO IMPACTO DO TLS NA PERFORMANCE DA COMUNICAÇÃO MQTT USANDO MOSQUITTO

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Marco Aurélio Spohn

CHAPECÓ
2026

Silva, Giovane Gonçalves da

AVALIAÇÃO DO IMPACTO DO TLS NA PERFORMANCE DA COMUNICAÇÃO MQTT USANDO MOSQUITTO / Giovane Gonçalves da Silva – 2026.

40 f.

Orientador: Marco Aurélio Spohn

Trabalho de Conclusão de Curso (Graduacão) – Universidade Federal da Fronteira Sul, Curso de Ciência da Computação, Chapecó, SC, 2026.

1. MQTT 2. Mosquitto 3. TLS 4. Performance 5. Avaliação I. Spohn, Marco Aurélio, orient. II. Universidade Federal da Fronteira Sul. III. Título.

GIOVANE GONÇALVES DA SILVA

AVALIAÇÃO DO IMPACTO DO TLS NA PERFORMANCE DA COMUNICAÇÃO MQTT USANDO MOSQUITTO

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Marco Aurélio Spohn

Este Trabalho de Conclusão de Curso foi avaliado e aprovado pela banca avaliadora em: 24/06/2026.

BANCA AVALIADORA

Marco Aurélio Spohn - UFFS

Andrei De Almeida Sampaio Braga - UFFS

Claunir Pavan - UFFS

DEDICATÓRIA

Dedico este trabalho a todos que aguentaram meu sumiço nos últimos meses — família, companheira e amigos. Vocês sabiam que ia dar certo antes de mim.

“A maioria das boas ideias é simples.”
(Linus Torvalds)

AGRADECIMENTOS

Agradeço ao professor Marco Aurélio Spohn pela orientação e paciência, principalmente com alguns sumiços ocasionais. À minha noiva e à minha família pelo apoio constante e por entenderem as ausências. Aos colegas que acompanharam essa jornada, pelas trocas e pela parceria.

RESUMO

O protocolo MQTT é amplamente empregado em aplicações de Internet das Coisas (IoT) devido à sua leveza e eficiência, características essenciais em dispositivos com recursos computacionais limitados. Entretanto, o protocolo não fornece mecanismos nativos de segurança robustos, o que torna comum a utilização do Transport Layer Security (TLS) para garantir confidencialidade, autenticidade e integridade na comunicação. Embora o TLS fortaleça a segurança, sua adoção pode introduzir sobrecarga computacional, especialmente durante o *handshake* e no processo contínuo de cifragem dos dados. Este trabalho investigou o impacto do uso de TLS no desempenho do broker Mosquitto, comparando cenários equivalentes com e sem criptografia, sob diferentes cargas de mensagens e quantidade de *clients*, mensurando latência, throughput, uso de CPU e uso de memória. Os resultados mostraram que o impacto do TLS no throughput é desprezível sob cargas moderadas, permanecendo abaixo de 1% até 75 *clients* simultâneos, mas se torna significativo em cargas elevadas, atingindo *overhead* de 9,9% em 100 *clients* e 12,8% em 110 *clients*, com sinais de instabilidade a partir de aproximadamente 85 *clients*. Na latência, o TLS apresentou *overhead* de 6 a 7% em volumes menores de mensagens, convergindo com o cenário sem TLS em volumes maiores. Já no uso de CPU e memória, o *overhead* se mostrou baixo em termos absolutos, com consumo de memória praticamente equivalente entre os dois cenários (159MB sem TLS e 162MB com TLS). Esses resultados fornecem evidências quantitativas que auxiliam na compreensão do trade-off entre segurança e desempenho em sistemas baseados em MQTT.

Palavras-Chave: MQTT, Mosquitto, TLS, Performance, Avaliação.

ABSTRACT

The MQTT protocol is widely used in Internet of Things (IoT) applications due to its lightweight design and efficiency, features that are crucial for devices with limited computational resources. However, MQTT does not provide robust native security mechanisms, making the use of Transport Layer Security (TLS) a common approach to ensure confidentiality, authenticity and message integrity. Although TLS enhances communication security, it may introduce computational overhead, particularly during the handshake and continuous data encryption processes. This work investigated the impact of TLS usage on the performance of the Mosquitto broker by comparing equivalent scenarios with and without encryption, under different message loads and numbers of clients, measuring latency, throughput, CPU usage and memory consumption. Results showed that the TLS impact on throughput is negligible under moderate loads, remaining below 1% up to 75 simultaneous clients, but becomes significant under high loads, reaching an overhead of 9.9% at 100 clients and 12.8% at 110 clients, with instability signs starting at approximately 85 clients. Regarding latency, TLS introduced an overhead of 6 to 7% for smaller message volumes, converging with the non-TLS scenario at larger volumes. CPU and memory usage showed low overhead in absolute terms, with nearly equivalent memory consumption between both scenarios (159MB without TLS and 162MB with TLS). These results provide quantitative evidence to support the understanding of the trade-off between security and performance in MQTT-based systems.

Keywords: MQTT, Mosquitto, TLS, Performance, Evaluation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACL – Access Control List

AES – Advanced Encryption Standard

CPU – Central Processing Unit

CSV – Comma-Separated Values

ECC – Elliptic Curve Cryptography

GCP – Google Cloud Platform

IOT – Internet of Things

JSON – JavaScript Object Notation

LWT – Last Will and Testament

MQTT – Message Queuing Telemetry Transport

QOS – Quality of Service

RAM – Random Access Memory

RFC – Request for Comments

RSA – Rivest–Shamir–Adleman

TCC – Trabalho de Conclusão de Curso

TCP – Transmission Control Protocol

TLS – Transport Layer Security

UFFS – Universidade Federal da Fronteira Sul

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização

A Internet das Coisas (IoT) vem ganhando espaço nas últimas décadas e já é uma realidade em diversos contextos, como automação industrial e residencial, monitoramento ambiental, agricultura de precisão e sistemas veiculares. Esses dispositivos, ao serem conectados à rede, permitem a coleta de métricas, o envio de dados e até a execução de ações remotas. No entanto, grande parte desses equipamentos é projetada para realizar tarefas específicas e, por isso, costuma dispor de recursos computacionais limitados, com restrições de processamento, memória e largura de banda (SONI; MAKWANA, 2017).

Para que a IoT seja escalável e economicamente viável, é essencial utilizar protocolos de comunicação leves, capazes de operar de forma eficiente mesmo em hardware simples. Entre as alternativas disponíveis, o CoAP (*Constrained Application Protocol*) se destaca por sua leveza, mas oferece mecanismos limitados de confiabilidade quando comparado às garantias de entrega disponíveis no MQTT. Já o AMQP (*Advanced Message Queuing Protocol*) fornece mecanismos robustos de persistência e roteamento, porém exige mais recursos computacionais. Nesse cenário, o MQTT (*Message Queuing Telemetry Transport*) se consolidou como uma solução intermediária: é leve, opera bem em ambientes limitados e oferece níveis de garantia de entrega, embora possua recursos nativos de segurança restritos (OASIS, 2019).

Assim como em qualquer sistema conectado à internet, a segurança é um ponto fundamental. O MQTT não foi projetado originalmente com foco na proteção das comunicações, oferecendo apenas mecanismos básicos de autenticação. Por isso, cabe ao *broker* incorporar ferramentas adicionais de segurança. Uma das principais ferramentas é o TLS (*Transport Layer Security*), que estabelece um canal seguro e encapsula os dados em uma camada criptográfica, garantindo confidencialidade, integridade e autenticação (RESCORLA, 2018). Ao incorporar essa camada, o MQTT aproxima-se dos níveis de segurança oferecidos por protocolos mais robustos como o AMQP, citado anteriormente, sem abrir mão de grande parte de sua leveza original, o que reforça a relevância de investigar o real impacto dessa combinação sobre o desempenho do protocolo.

A adoção do TLS, entretanto, introduz um custo computacional adicional. O processo de *handshake*, responsável pela troca inicial de chaves, envolve operações criptográficas mais pesadas e tende a ser o momento mais impactado. Além disso, a

cifragem contínua dos dados durante a sessão pode aumentar o uso de CPU, memória e latência. Esse *trade-off* entre segurança e desempenho motivou a investigação conduzida neste trabalho, cujos resultados são apresentados e discutidos no capítulo 5.

1.2 Problema de Pesquisa

É um fato conhecido que o TLS introduz uma sobrecarga computacional. Por combinar criptografia assimétrica durante o *handshake* com criptografia simétrica na transmissão dos dados, o protocolo exige mais processamento, especialmente no momento em que a conexão é estabelecida (RESCORLA, 2018). Em dispositivos com *hardware* limitado, esse custo se torna ainda mais significativo, fazendo com que aplicações IoT, que normalmente são restritas em processamento, memória e energia, sejam as mais afetadas.

Embora existam diversos trabalhos que analisam o desempenho do MQTT com TLS, os resultados apresentados não são diretamente comparáveis entre si, pois cada estudo utiliza diferentes *hardwares*, *brokers*, cargas de trabalho e metodologias de medição (SEOANE et al., 2021). Além disso, apenas parte desses estudos avalia especificamente o *broker* Mosquitto, o que dificulta uma análise clara e padronizada do impacto real da camada TLS nesse *broker* em particular, como discutido no capítulo 3.

Diante desse cenário, este trabalho buscou responder à seguinte questão: qual é o impacto da utilização do protocolo TLS no desempenho do *broker* Mosquitto em diferentes níveis de carga, considerando as métricas de desempenho latência, *throughput*, uso de CPU e uso de memória? A resposta a essa questão, construída a partir da execução de testes controlados, é apresentada e discutida ao longo deste trabalho.

1.3 Objetivo Geral

Avaliar o impacto do uso de TLS no desempenho do *broker* Mosquitto, comparando diferentes cenários de carga em configurações com e sem o uso do protocolo TLS.

1.4 Objetivos Específicos

- Analisar como a ativação do TLS afeta a latência e o *throughput* na comunicação MQTT utilizando o *broker* Mosquitto.
- Investigar o impacto do TLS no consumo de recursos do *broker*, especialmente uso de CPU e memória, sob diferentes níveis de carga.
- Comparar o comportamento do Mosquitto em cenários com e sem TLS, identificando variações de desempenho associadas à criptografia.
- Avaliar de forma quantitativa a relação entre aumento de carga e degradação de desempenho em cenários com TLS.

1.5 Justificativa

O *Message Queuing Telemetry Transport* (MQTT) é amplamente adotado na Internet das Coisas devido à sua leveza e eficiência, características essenciais para dispositivos que operam com recursos limitados (SONI; MAKWANA, 2017). No entanto, a necessidade de garantir segurança nas comunicações torna o uso do TLS um requisito importante em muitos cenários. A adição dessa camada, contudo, pode impactar o desempenho do sistema, especialmente em ambientes restritos de hardware.

O Mosquitto é um dos *brokers* MQTT mais utilizados atualmente, e compreender o *trade-off* entre segurança e desempenho é fundamental para o desenvolvimento de soluções eficientes e seguras. Os resultados obtidos neste trabalho, detalhados no capítulo 5, fornecem evidências quantitativas que podem auxiliar desenvolvedores e engenheiros a decidir sobre a solução mais adequada para cada aplicação, considerando os requisitos específicos de segurança e demanda computacional.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será apresentado o *broker* Mosquitto, uma ferramenta comumente usada para a implementação do protocolo. Também serão apresentados o protocolo TLS e seus mecanismos fundamentais de segurança.

2.1 O protocolo MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de mensagens leve, que utiliza o paradigma *publish/subscribe*, projetado para comunicação leve, em ambientes onde os recursos de hardware e a largura de banda são restritos. O MQTT se situa na camada de aplicação do modelo TCP/IP, utilizando como protocolo de transporte o TCP, sendo utilizado principalmente em aplicações da Internet das Coisas (IoT) para troca de dados de forma simples, leve e escalável (THAVAMANI; SINTHUJA, 2021; OASIS, 2019).

A arquitetura do MQTT possui dois elementos centrais: os *clients* e o *broker*. Os *clients* podem atuar tanto como *publishers* como *subscribers*, enquanto o *broker* atua como um servidor central que intermedeia toda a comunicação: cabe a ele receber as mensagens publicadas em um tópico e redistribuí-las a todos os *clients* que assinaram esse mesmo tópico. Ao concentrar essa função de intermediação, o *broker* desacopla *publishers* de *subscribers* tanto no espaço quanto no tempo, permitindo que a comunicação ocorra entre os dispositivos emissores e receptores sem que os *clients* precisem conhecer diretamente os demais participantes do sistema, apenas os tópicos em que desejam publicar ou assinar (SONI; MAKWANA, 2017). Um exemplo prático ilustra esse funcionamento: em uma estufa automatizada, um sensor de temperatura publica periodicamente suas leituras no tópico `sensores/temperatura`. Uma aplicação de monitoramento, inscrita nesse mesmo tópico, recebe cada atualização automaticamente por meio do *broker* e a utiliza para exibir a temperatura em um painel em tempo real, disparando um alerta caso o valor ultrapasse um limite pré-definido. Nesse cenário, o sensor não precisa conhecer a aplicação de monitoramento, nem esta precisa se comunicar diretamente com o sensor, pois ambos interagem apenas por meio dos tópicos publicados no *broker*.

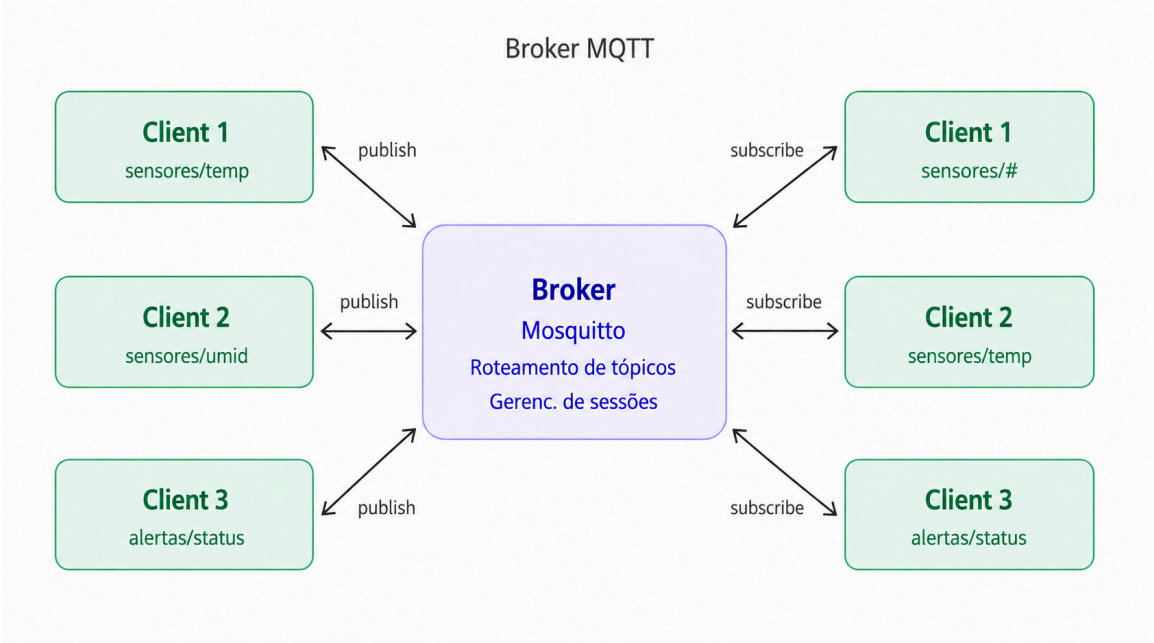

Figura 2.1 – Arquitetura *publish/subscribe* do protocolo MQTT.

A comunicação entre *clients* e o *broker* é estabelecida geralmente sobre o protocolo TCP, em uma conexão persistente e bidirecional. Essa abordagem possibilita que um mesmo *client* envie e receba mensagens durante a mesma sessão. O MQTT adota uma estrutura hierárquica de tópicos, na qual os nomes são organizados por níveis separados por barras ("/"), permitindo o uso de caracteres curinga (*wildcards*) como "+"e "#"para assinatura flexível de múltiplos tópicos (OASIS, 2019).

O MQTT possui recursos para controlar as sessões, possibilitando a utilização de sessões limpas (*clean session*) e sessões persistentes, também possui recursos de retenção de mensagens (*retained messages*), e mensagem de último desejo (*Last Will and Testament – LWT*), que servem para aumentar a confiabilidade da comunicação em casos onde ocorre uma desconexão inesperada (OASIS, 2019; THAVAMANI; SINTHUJA, 2021).

Permitindo níveis diferentes de confiabilidade de entrega das mensagens, o MQTT conta com três Qualidades de Serviço (*Quality of Service – QoS*): **QoS 0** (*at most once*), no qual a mensagem é enviada sem confirmação de entrega; **QoS 1** (*at least once*), que possui garantia de entrega da mensagem através da confirmação de recebimento (*PUBACK*) por parte do *client*, podendo gerar duplicações; e **QoS 2** (*exactly once*), que possui 4 etapas para garantir entrega única e prevenir duplicidade (*PUBLISH* $\rightarrow$ *PUBREC* $\rightarrow$ *PUBREL* $\rightarrow$ *PUBCOMP*) (OASIS, 2019; LIGHT, 2017).

Graças à leveza das mensagens, simplicidade na utilização e baixo uso de recursos, o MQTT é hoje uma escolha popular para sistemas embarcados e redes com baixa largura de banda. No entanto, essas mesmas características exigem o uso

de mecanismos adicionais de segurança e autenticação, como o TLS, para assegurar confidencialidade e integridade nas comunicações entre *clients* e o *broker* (LIGHT, 2017; SONI; MAKWANA, 2017).

2.1.1 O broker Mosquitto

O Eclipse Mosquitto é uma implementação de código aberto do protocolo MQTT, desenvolvido por Roger Light em 2010, e atualmente mantida pela Eclipse Foundation (LIGHT, 2017). É amplamente reconhecido pela eficiência e compatibilidade com diversos dispositivos e sistemas operacionais.

A arquitetura do Mosquitto segue o paradigma *publish/subscribe* do MQTT, atuando como intermediário na comunicação entre os *clients*, sejam eles *publishers* ou *subscribers*. O *broker* atua como ponto central de comunicação, recebendo todas as mensagens dos *publishers* e encaminhando-as aos *subscribers* apropriados (OASIS, 2014).

Ele é responsável pelo gerenciamento das conexões e sessões persistentes, pela distribuição de mensagens e pelo controle de acesso. Também implementa os três níveis de QoS definidos pelo protocolo MQTT.

Indo além das especificações do protocolo, o Mosquitto oferece suporte a *retained messages* e ao mecanismo de LWT (*Last Will and Testament*). Ainda, conta com controle de acesso baseado em ACL (*Access Control List*) e autenticação básica, além de mecanismos de persistência local para garantir a entrega de mensagens em cenários de desconexão ou falha de comunicação.

2.2 O protocolo TLS

O ***Transport Layer Security (TLS)*** é o protocolo mais usado para criptografia na internet, utilizando processos criptográficos combinados, garantindo a confidencialidade, integridade e autenticação na comunicação entre agentes (GRIGORIK, 2013). O TLS atua sobre o protocolo TCP, entre as camadas de Transporte e Aplicação no modelo TCP/IP, funcionando como uma camada adicional de segurança (GRIGORIK, 2013). O TLS criptografa os dados trafegados de um agente a outro, protegendo o conteúdo contra interceptação e garantindo confidencialidade.

Para garantir essa confidencialidade, o TLS utiliza criptografia simétrica para cifrar dados, criptografia assimétrica para o processo de *handshake* e troca de chaves,

e funções *hash* para assegurar integridade. Apenas os agentes envolvidos na comunicação têm acesso às chaves que permitem descriptografar o conteúdo. A Figura 2.2 apresenta a sequência de mensagens do *handshake* TLS.

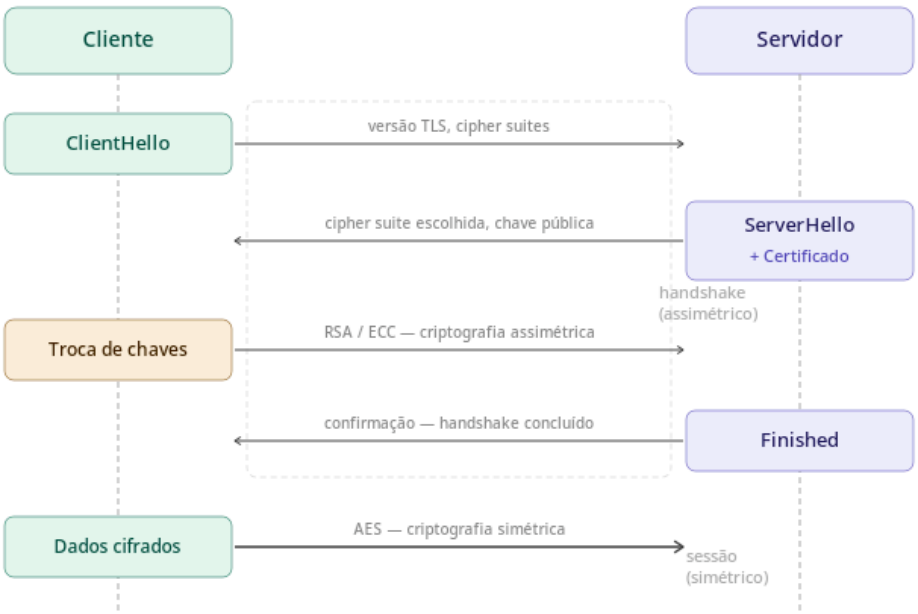

Figura 2.2 – Sequência de mensagens do *handshake* TLS.

No contexto do MQTT, a utilização do TLS é fundamental para troca segura de mensagens, principalmente quando os dados transmitidos são sensíveis. Contudo, adicionar essa camada de segurança gera impactos de performance, como aumento da latência, maior consumo de CPU e memória, e sobrecusto no *handshake* (HiveMQ, 2023).

2.2.1 Criptografia

A criptografia é o processo de transformar um texto legível em um texto cifrado por meio de algoritmos matemáticos, garantindo que apenas agentes autorizados possam recuperar o conteúdo original. O TLS utiliza criptografia para proteger dados mesmo em caso de interceptação (RESCORLA, 2018).

Neste protocolo, a criptografia simétrica é usada para cifrar os dados durante a sessão — sendo o AES um dos algoritmos mais comuns (Cloudflare, 2019). A criptografia assimétrica é usada no *handshake* para autenticação e troca segura da

chave simétrica, e envolve algoritmos mais robustos e custosos, como RSA ou ECC (HiveMQ, 2023).

A criptografia também garante integridade através de funções de *hash* como SHA-256. Por envolver operações intensivas, o TLS adiciona custos computacionais à comunicação MQTT, aumentando o tempo de conexão e uso de recursos do sistema.

2.3 Impacto de TLS no MQTT

Utilizar o TLS junto com o MQTT torna a comunicação mais segura, garantindo autenticação e confidencialidade das informações. No entanto, essa segurança adicional adiciona um custo computacional decorrente do processamento nas operações de troca de chaves, criptografia contínua e aumento no tamanho dos pacotes.

Estudos anteriores indicam que o uso do TLS aumenta o consumo de CPU e memória, sendo esse impacto mais pronunciado em ambientes com recursos limitados, onde o custo do *handshake* e da cifragem contínua se torna mais significativo em relação à capacidade disponível. Em contrapartida, quando o *hardware* é mais robusto, o impacto tende a ser menor (GAVRIILIDIS; HALKIDIS; PETRIDOU, 2025).

2.4 Benchmarks e testes

Benchmarking é um método de análise de desempenho que permite avaliar eficiência, escalabilidade e estabilidade de aplicações sob diferentes cargas (MISHRA; MISHRA; KERTESZ, 2021). Diversos estudos analisam como brokers MQTT se comportam variando QoS, persistência, TLS e outros fatores (DIZDAREVIC; MICHALKE; JUKAN, 2023; EMQX, 2023). Para testes automatizados, ferramentas como *emqtt-bench*, *MQTT-stresser* e *JMeter* são usadas (HiveMQ, 2022; Altoros Labs, 2023). Essas métricas fundamentam a análise experimental desenvolvida neste trabalho.

2.4.1 Tecnologias utilizadas e seus papéis no trabalho

A Tabela 2.1 resume as principais tecnologias que compõem o ambiente analisado, relacionando cada uma delas com seu papel dentro do escopo deste estudo.

Tecnologia	**Papel no Trabalho**
MQTT	Protocolo de comunicação utilizado para avaliar o impacto do uso de TLS no tráfego de mensagens.
Mosquitto	*Broker* MQTT avaliado nos experimentos, devido à sua leveza e ampla adoção em ambientes IoT.
TLS	Camada de segurança cuja ativação foi comparada em termos de impacto de desempenho.
Google Cloud	Infraestrutura utilizada para a execução dos ambientes de teste equivalentes.
Terraform	Ferramenta de provisionamento de infraestrutura utilizada para criar e configurar as máquinas virtuais de forma reproduzível.
Ansible	Ferramenta de automação utilizada para configurar os ambientes de teste de forma padronizada.
emqtt-bench	Ferramenta utilizada para geração de carga e coleta de métricas de *throughput*.
Python + Paho-MQTT	Scripts utilizados para medição de latência ponto a ponto durante a execução dos testes.
vmstat	Ferramenta utilizada para coleta de métricas de uso de CPU e memória nos brokers durante os testes.

Tabela 2.1 – Tecnologias empregadas e seus papéis no desenvolvimento deste trabalho.

3. TRABALHOS RELACIONADOS

Há diversos estudos que analisam o desempenho do protocolo MQTT sob diferentes configurações, especialmente no que diz respeito ao impacto da adição do TLS como camada de segurança. De modo geral, esses trabalhos se concentram em ambientes de Internet das Coisas (IoT), nos quais o MQTT é amplamente empregado devido ao seu baixo consumo de recursos.

Seoane et al. (2021) realizaram uma análise comparativa entre CoAP e MQTT com suporte a TLS em dispositivos IoT, sem especificar um broker único como foco principal da avaliação. Os autores mostram que a adição da camada de segurança aumenta a latência e reduz o *throughput*, especialmente em ambientes onde o hardware é limitado. O estudo também evidencia que o impacto do TLS é mais perceptível em cenários restritos, reforçando a necessidade de avaliar a viabilidade do uso de criptografia em aplicações embarcadas (SEOANE et al., 2021).

Gavriilidis et al. (2025) avaliaram o uso do MQTT com TLS em dispositivos IoT, utilizando Raspberry Pi 4B como plataforma de testes e o Mosquitto como broker. Os autores observaram que o maior custo computacional ocorre no momento do *handshake* TLS, devido às operações criptográficas assimétricas envolvidas na troca de chaves. Durante o tráfego contínuo, o impacto é menor, mas ainda perceptível em termos de consumo de CPU. O estudo também compara diferentes *cipher suites* e mostra que ChaCha20_Poly1305 tende a apresentar melhor desempenho em ambientes sem fio (GAVRIILIDIS; HALKIDIS; PETRIDOU, 2025).

Prantl et al. (2021) analisaram o impacto do TLS no desempenho do MQTT utilizando um microcontrolador ESP8266 como *client* e o Mosquitto como *broker*. Os autores avaliaram *throughput*, tempo de estabelecimento de conexão e eficiência energética sob diferentes condições de rede. Os resultados indicam que o impacto negativo do TLS depende das condições de rede e da frequência de reconexão, sendo mais perceptível em cenários com degradação de rede ou com reconexões frequentes, enquanto em condições normais de operação o impacto no tráfego contínuo mostrou-se limitado (PRANTL et al., 2021).

Mishra et al. (2021) realizaram testes de estresse em seis implementações de *brokers* MQTT — Mosquitto, ActiveMQ, HiveMQ, Bevywise, VerneMQ e EMQ X —, avaliando métricas como taxa de mensagens, uso de CPU e latência. Os resultados mostram que o desempenho varia significativamente conforme a arquitetura de cada *broker*: o Mosquitto apresentou melhor desempenho geral na maioria das métricas, enquanto o ActiveMQ destacou-se em escalabilidade por sua implementação multithread (MISHRA; MISHRA; KERTESZ, 2021).

Os estudos analisados indicam que há um aumento do custo computacional ao se adicionar o TLS, principalmente durante o estabelecimento da conexão, que é o momento em que se exige mais processamento, mas no tráfego contínuo na conexão também é perceptível.

Entretanto, ainda que seja notório esse impacto, os trabalhos divergem entre si nos ambientes utilizados nos testes: enquanto alguns autores utilizaram servidores ou *laptops* como *broker*, outros utilizaram dispositivos IoT como *clients*, tornando a comparação direta dos resultados obtidos por eles difícil.

Dada essa diversificação de abordagens de teste, evidencia-se a necessidade de uma análise quantitativa e mais precisa em relação ao impacto em *brokers* específicos, como o Mosquitto.

4. METODOLOGIA

4.1 Estrutura do Ambiente de Testes

Neste trabalho foram utilizados três ambientes de teste no Google Cloud Platform: duas instâncias executando o *broker* Mosquitto e uma instância dedicada à geração de carga e coleta de métricas. As três máquinas virtuais são do tipo e2-medium, contendo 2 vCPUs e 4 GB de memória RAM, executando o sistema operacional Debian 11.11 (Bullseye), e foram provisionadas com Terraform e configuradas com Ansible, garantindo reprodutibilidade do ambiente.

As instâncias de *broker* são idênticas quanto à instalação e configuração do Mosquitto, diferenciando-se apenas pela ativação do TLS: na primeira, o *broker* operou na porta padrão 1883 sem criptografia, enquanto na segunda foi configurado na porta 8883 com suporte ao protocolo TLS. O certificado TLS foi emitido via Let's Encrypt utilizando o Certbot, com *hostname* válido provido pelo serviço No-IP, eliminando variáveis associadas ao uso de certificados autoassinados. Na configuração do ambiente também foram definidos limites de utilização dos recursos da máquina, de forma a evitar sobrecarga no sistema operacional, que poderia prejudicar a continuidade dos testes ou distorcer os resultados. Dessa forma, o Mosquitto pôde negar serviço em caso de saturação, sem comprometer a operação da máquina virtual, permitindo a recuperação do ambiente.

Essa estrutura garantiu paridade entre os ambientes e permitiu isolar o impacto causado exclusivamente pela camada de segurança, possibilitando uma comparação direta do desempenho entre os dois cenários. A Figura 4.1 ilustra a arquitetura do ambiente de testes, e a Tabela 4.1 apresenta as versões dos componentes utilizados.

Os arquivos de configuração do Terraform e do Ansible utilizados para provisionar e configurar o ambiente, bem como os scripts empregados na execução dos testes e na análise dos dados, estão disponíveis publicamente em um repositório no GitHub[1], possibilitando a replicação integral do experimento.

[1] Disponível em: <https://github.com/thegiovanesilva/mqtt-tls-performance-evaluation>

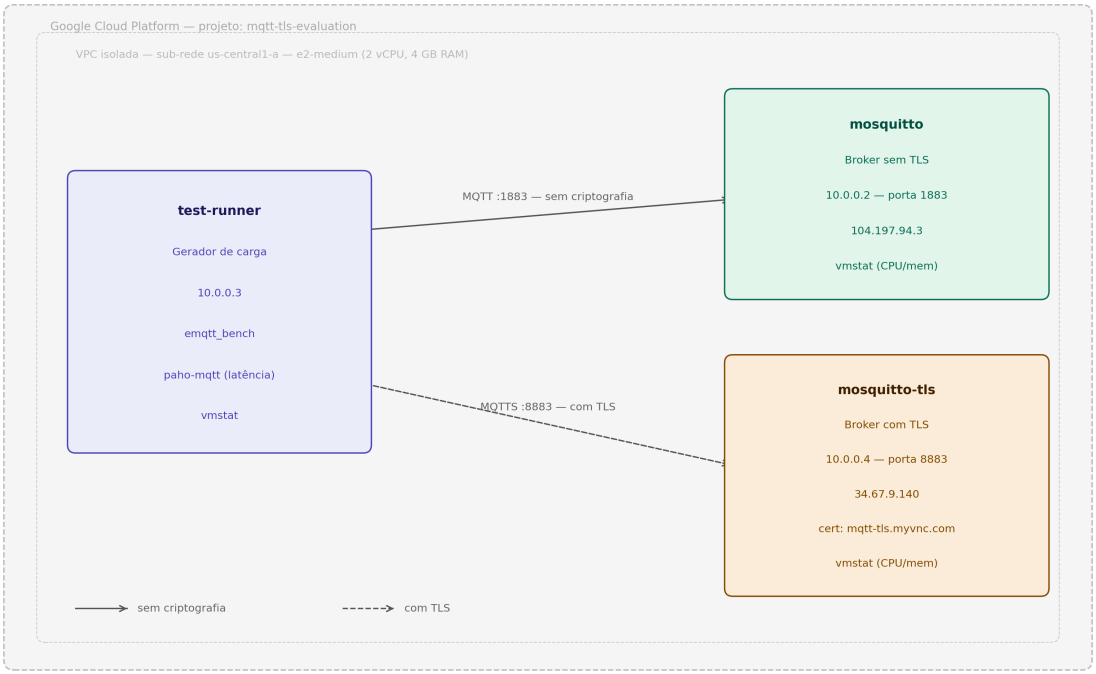

Figura 4.1 – Arquitetura do ambiente de testes no Google Cloud Platform.

Componente	**Versão**
Mosquitto	2.0.11
Protocolo MQTT	3.1.1
TLS	1.3 (OpenSSL 1.1.1w)
emqtt-bench	0.6.0
Python	3.9.2
paho-mqtt	2.1.0
Erlang/OTP	28
vmstat	procps-ng 3.3.17
Sistema operacional	Debian 11.11 (Bullseye)
Kernel	5.10.0-44-cloud-amd64

Tabela 4.1 – Versões dos componentes utilizados no ambiente de testes.

4.2 Métricas de Desempenho e Coleta de Dados

As métricas analisadas foram latência, taxa de transferência (*throughput*), uso de CPU e uso de memória, medidas em ambos os cenários sob diferentes níveis de carga. Essas métricas permitem avaliar o custo computacional associado à

ativação do TLS no *broker*, bem como observar o impacto na comunicação MQTT à medida que a carga aumenta.

Os testes foram repetidos 15 vezes por cenário, organizados em 5 baterias de execução, totalizando 75 amostras por combinação de carga. Essa quantidade foi considerada suficiente para reduzir a influência de oscilações pontuais e permitir uma análise consistente das métricas coletadas.

A geração de carga e a coleta de *throughput* foram realizadas com a ferramenta *emqtt-bench*. A latência foi medida separadamente por meio de scripts em Python utilizando a biblioteca *paho-mqtt*, com *timestamps* embutidos nas mensagens para cálculo do tempo de entrega ponto a ponto, essa abordagem foi adotada pois o *emqtt-bench* não reporta latência de forma confiável. O uso de CPU e memória foi monitorado com a ferramenta *vmstat*, executada continuamente nos *brokers* durante toda a coleta.

4.3 Procedimentos de Teste

Os testes foram realizados de forma controlada, variando-se o número de *clients* simultâneos mantendo o intervalo de publicação fixo em 200ms. Intervalos menores foram avaliados em testes preliminares, mas causavam saturação do *broker* e *timeouts*, comprometendo a estabilidade dos resultados. O tamanho das mensagens foi fixado em 256B para isolar o efeito do número de *clients*; a análise de diferentes tamanhos de mensagem foi deixada como trabalho futuro.

Foram executadas duas baterias de testes. A bateria principal avaliou cargas de 1 a 100 *clients* nos intervalos {1, 5, 10, 25, 50, 75, 100}, com duração de 30 segundos por repetição. A bateria de saturação avaliou a faixa de 75 a 110 *clients* nos intervalos {75, 80, 85, 90, 95, 100, 105, 110}, com o objetivo de caracterizar com maior precisão o ponto a partir do qual o *broker* com TLS apresenta instabilidade, identificado na bateria principal a partir de 100 *clients*.

Em cada teste, o fluxo seguiu a ordem: inicialização do *broker*, início da coleta de métricas com *vmstat*, execução dos testes com o número de *clients* definido para o cenário, e finalização controlada antes da repetição. Os dados de *throughput* e latência foram armazenados em formato JSON com *timestamp*, e os dados de CPU e memória em formato CSV, para posterior análise com scripts Python.

4.4 Cronograma de Execução

O cronograma foi organizado em semanas, contendo a implantação do ambiente de testes, a execução das medições, a análise dos resultados e a redação final do trabalho. As atividades estão distribuídas sequencialmente, permitindo que os experimentos fossem realizados de forma reprodutível e controlada.

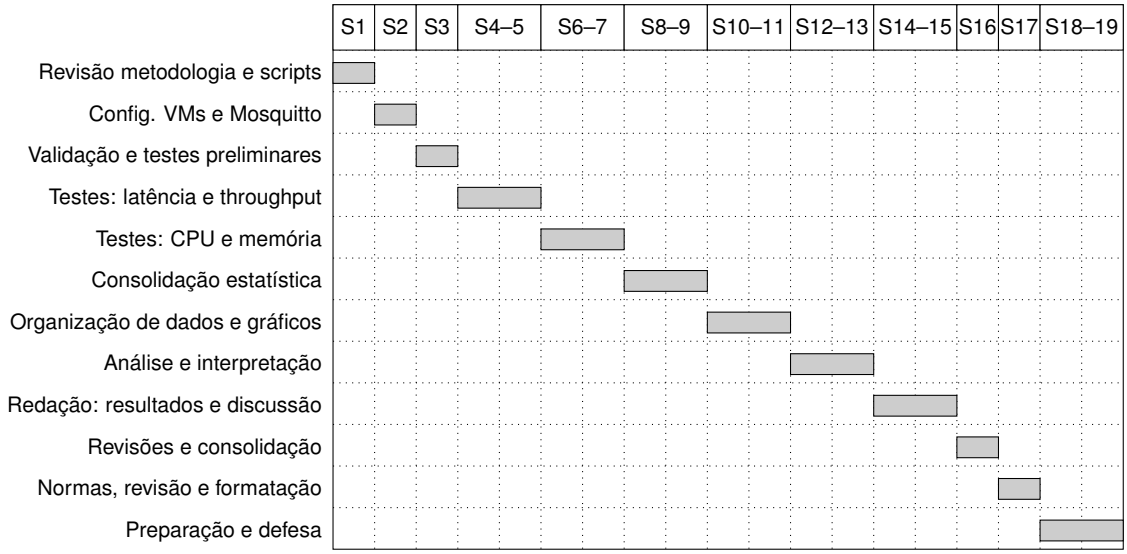

Figura 4.2 – Cronograma de execução do TCC II no semestre 2026/1.

4.5 Considerações Finais

A metodologia descrita neste capítulo estabeleceu um ambiente replicável e controlado, adequado para avaliar de forma isolada o impacto do TLS no desempenho do *broker* Mosquitto. A forma como os ambientes foram estruturados, as métricas selecionadas e os procedimentos de teste definidos permitiram isolar a influência da camada de segurança das demais variáveis, possibilitando a comparação direta entre os dois cenários. O cronograma organizou a execução dos testes em etapas sequenciais, garantindo condições adequadas para a coleta e posterior análise dos dados.

5. RESULTADOS E DISCUSSÃO

Este capítulo apresenta e discute os resultados obtidos com a execução das baterias de teste descritas na Metodologia, realizando um comparativo de desempenho entre os dois *brokers* configurados, com TLS ativo e sem TLS ativo. Os resultados são apresentados em quatro seções: *throughput*, ponto de saturação, latência e uso de CPU e memória.

5.1 Throughput

A Figura 5.1 apresenta o *throughput* médio obtido em cada nível de carga para os dois cenários. Até 75 *clients* simultâneos, o desempenho entre os dois *brokers* foi praticamente equivalente, com o *broker* com TLS apresentando *overhead* inferior a 1% nessa faixa. Isso indica que, sob cargas moderadas, o custo imposto pela cifragem do TLS 1.3 não apresenta impacto perceptível na capacidade de processamento do *broker*.

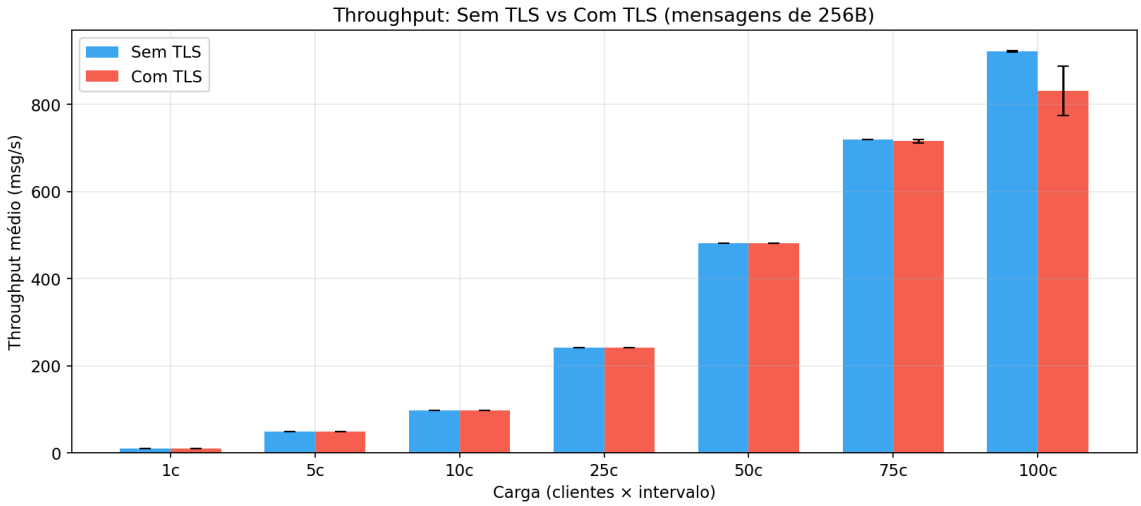

Figura 5.1 – Throughput médio por nível de carga: cenários sem TLS e com TLS.

A partir de 100 *clients*, o comportamento dos dois cenários diverge consideravelmente. O *broker* sem TLS manteve *throughput* médio de 922,11 msg/s com desvio padrão de 1,73 msg/s, enquanto o *broker* com TLS registrou 830,75 msg/s com desvio padrão de 56,79 msg/s, resultando em um *overhead* de 9,9%. A Figura 5.2 evidencia essa diferença na variabilidade. Enquanto o desvio padrão sem TLS se manteve baixo em toda a faixa avaliada, o desvio padrão do *broker* com TLS cresceu abruptamente

a partir dos 75 *clients*, atingindo valores que indicam comportamento instável nessa faixa de carga.

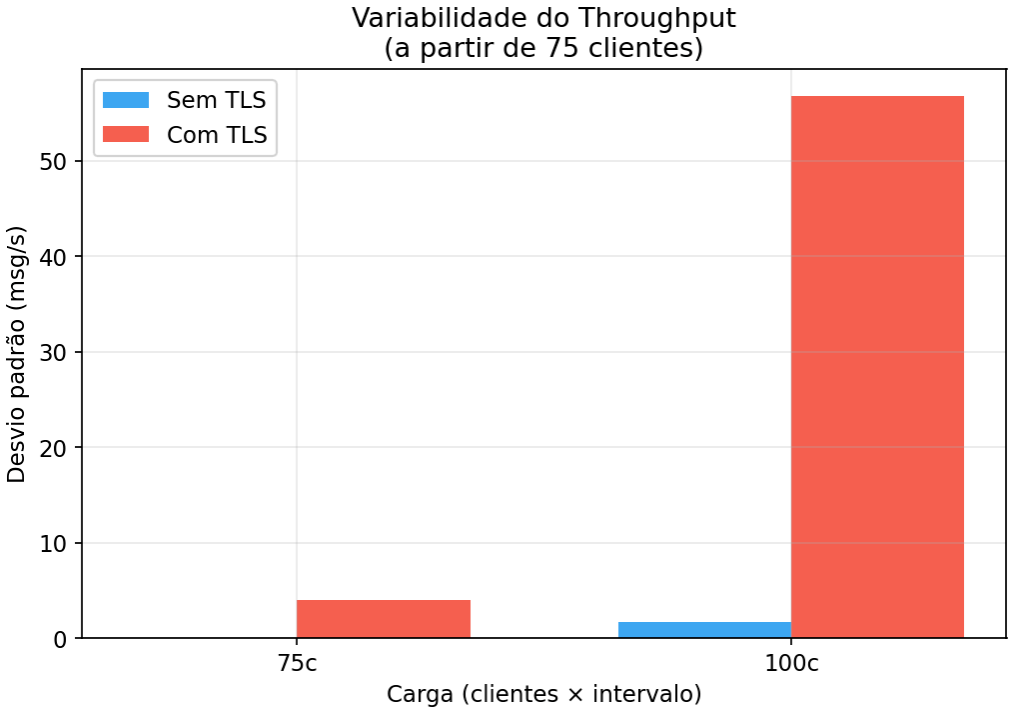

Figura 5.2 – Variabilidade do *throughput* a partir de 75 *clients* simultâneos.

A Figura 5.3 detalha a progressão do *overhead* do TLS no *throughput* a partir de 25 *clients*, ponto em que as diferenças começaram a aparecer. O impacto permaneceu abaixo de 1% até 75 *clients* e cresceu de forma abrupta em 100 *clients*, indicando que a degradação não ocorre de forma linear e se concentra próxima desse nível de carga. As possíveis causas são discutidas na seção de CPU e memória.

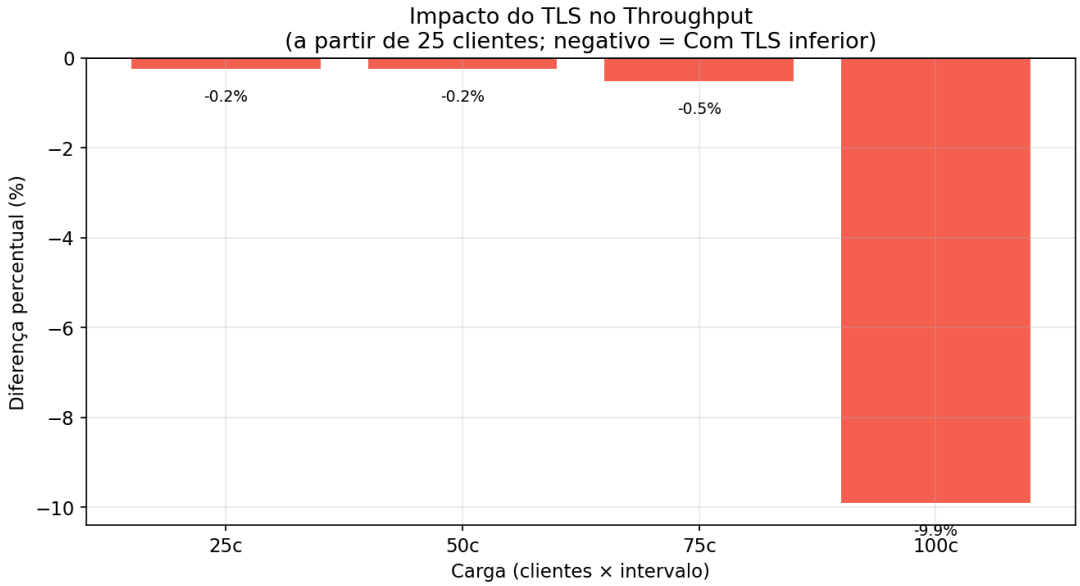

Figura 5.3 – Overhead do TLS no *throughput* a partir de 25 *clients* simultâneos.

Esses resultados estão alinhados com o observado por Gavriilidis et al. (2025) e Prantl et al. (2021), que identificaram impacto mais pronunciado do TLS em situações de maior demanda computacional. Nesses cenários, o custo da criptografia torna-se mais perceptível à medida que o sistema se aproxima do seu limite de capacidade.

5.2 Ponto de Saturação

A instabilidade observada em 100 *clients* na bateria principal motivou a execução de uma bateria de saturação, avaliando a faixa de 75 a 110 *clients* com intervalos de 5 *clients*. Os resultados indicam que o *broker* com TLS se manteve estável até 80 *clients*, com desvio padrão próximo de 1 msg/s. A partir de 85 *clients*, o desvio padrão aumentou para cerca de 25 msg/s, crescendo progressivamente até 73 msg/s em 110 *clients*, enquanto o *broker* sem TLS permaneceu estável e crescente em toda a faixa avaliada.

O *overhead* de *throughput* seguiu a mesma progressão: 0,4% em 75 *clients*, 0,7% em 80 *clients*, 2,3% em 85 *clients*, 6,4% em 90 *clients*, 8,2% em 95 *clients*, chegando a 12,8% em 110 *clients*. A partir de aproximadamente 90 *clients*, o *throughput* com TLS estabilizou em torno de 840 a 870 msg/s, deixando de acompanhar o crescimento observado no *broker* sem TLS. A alta variância observada nessa faixa é compatível com um cenário de saturação e não necessariamente com uma falha metodológica. Esse comportamento é esperado quando o sistema passa a operar próximo

de seus limites de processamento, tornando o desempenho menos previsível. Nas condições testadas, utilizando mensagens de 256B, intervalo de 200ms e hardware e2-medium, foram observados sinais de instabilidade a partir de aproximadamente 85 *clients*, com *throughput* médio em torno de 720 msg/s nesse cenário. Esse padrão se repetiu de forma consistente ao longo das cinco baterias de execução, o que reforça tratar-se de um comportamento característico dessa faixa de carga sob TLS, e não de uma oscilação pontual.

O ambiente de testes utilizou limites de recursos configurados na VM do *broker*, conforme descrito na Seção 4.1. Vale registrar que o uso de CPU e memória do *test-runner*, responsável pela geração da carga e pela execução das operações criptográficas do lado client nas conexões TLS, não foi monitorado durante os experimentos. Embora a associação entre essa instabilidade e a ativação do TLS esteja bem caracterizada pelos dados coletados, este trabalho não isola a parcela dessa degradação atribuível ao *broker* daquela eventualmente introduzida pelo próprio gerador de carga, distinção retomada na Seção 5.4.

5.3 Latência

A latência foi medida com um único *client* publicando mensagens sequencialmente, com *timestamps* embutidos nas mensagens para cálculo do tempo de entrega ponto a ponto. Essa abordagem foi adotada porque múltiplos *clients* simultâneos impedem o isolamento confiável da latência individual e porque o *emqtt-bench*, embora disponibilize um recurso de rastreamento de latência (*QoE tracking*), o exporta apenas de forma agregada, por meio de histogramas via Prometheus com *buckets* fixos (1, 5, 10, 25, 50, 100, 500, 1000ms, entre outros) (EMQX, 2026). Como as latências observadas neste ambiente situam-se na casa de poucos milissegundos, essa granularidade não permitiria distinguir as diferenças de frações de milissegundo entre os cenários com e sem TLS, além de exigir infraestrutura adicional de coleta (Prometheus e API REST) não utilizada nas demais métricas deste trabalho. Por esse motivo, optou-se por medir a latência ponto a ponto diretamente a partir dos *timestamps* brutos, permitindo o cálculo exato de médias e percentis sem a perda de precisão introduzida pela agregação em *buckets*.

A Figura 5.4 apresenta a latência média de entrega para cada volume avaliado. O *broker* com TLS apresentou latência superior nos volumes menores, com *overhead* de aproximadamente 6 a 7% em 200, 500 e 1000 mensagens. Em termos absolutos, a diferença foi de frações de milissegundo, aproximadamente 1,94ms sem

TLS contra 2,08ms com TLS em 1000 mensagens. Em 2000 e 3000 mensagens, a diferença reduziu-se e inverteu-se levemente, o que não permite afirmar vantagem consistente para nenhum dos cenários nesses volumes.

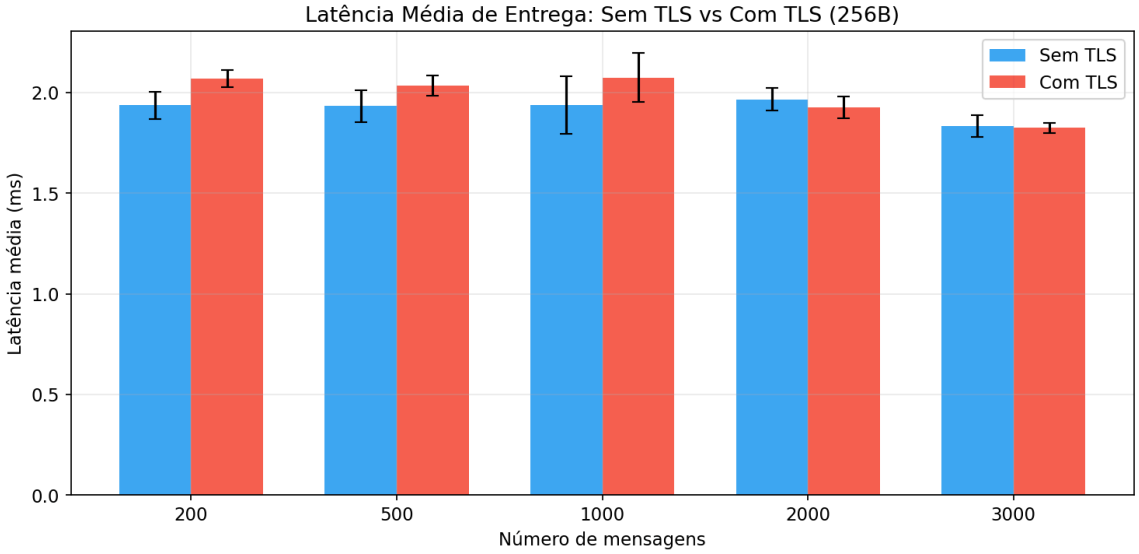

Figura 5.4 – Latência média de entrega por volume de mensagens (256B).

Além da latência média, é relevante observar o comportamento nos percentis mais altos da distribuição, que capturam os piores casos de entrega e são mais sensíveis a variações pontuais de desempenho. O percentil 95 (p95) indica o valor de latência abaixo do qual se concentram 95% das mensagens observadas, sendo uma métrica mais representativa do pior cenário típico do que a simples média. A Figura 5.5 apresenta a latência no percentil 95. O padrão observado é semelhante ao da latência média, com *overhead* leve do TLS nos volumes menores e convergência nos volumes maiores, confirmando que a diferença não está concentrada em eventos isolados.

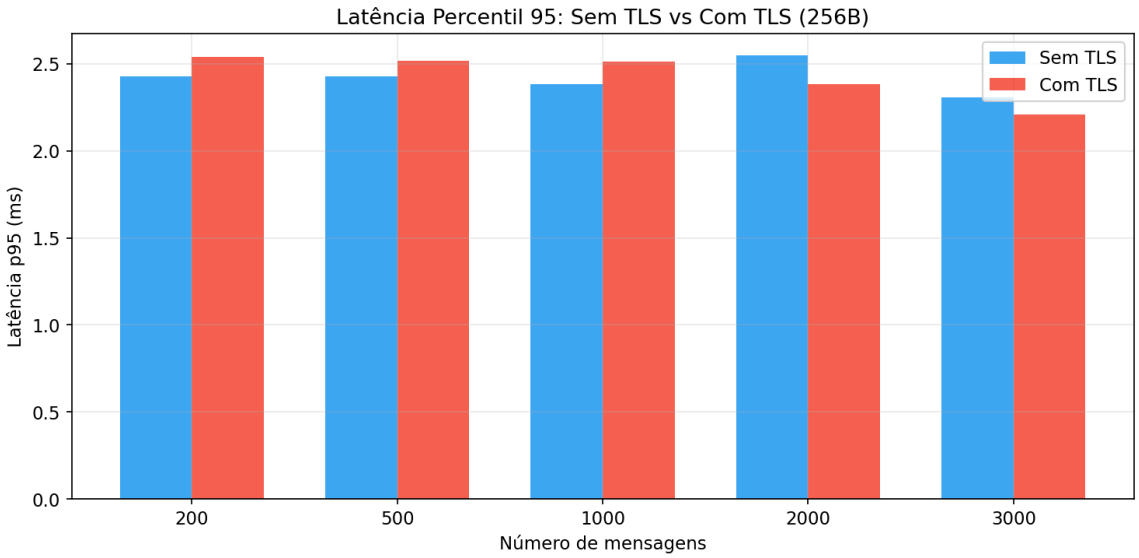

Figura 5.5 – Latência no percentil 95 por volume de mensagens (256B).

A Figura 5.6 apresenta a distribuição nos percentis p50, p95 e p99 para 1000 mensagens. O *broker* com TLS apresentou valores ligeiramente superiores em todos os percentis, com a diferença mantendo-se consistente ao longo da distribuição. Isso indica que o *overhead* de latência introduzido pelo TLS é estável e previsível nas condições avaliadas.

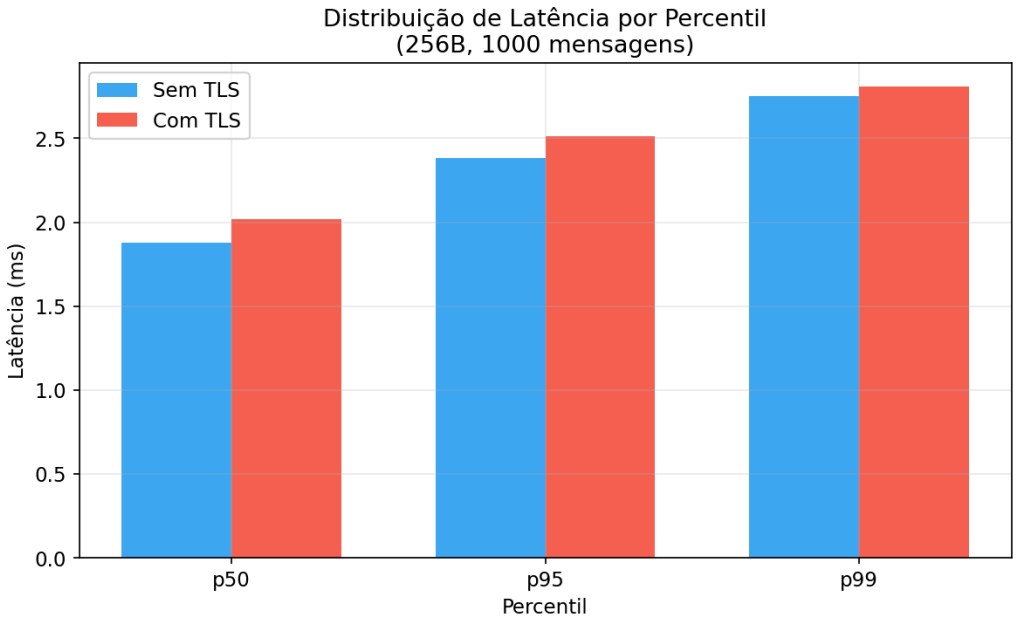

Figura 5.6 – Distribuição de latência por percentil (256B, 1000 mensagens).

5.4 Uso de CPU e Memória

A Figura 5.7 apresenta o uso médio de CPU nos *brokers* durante os testes. O *broker* com TLS apresentou maior consumo em espaço de usuário, aproximadamente 1,7%, contra 1,2% do *broker* sem TLS. Em espaço de sistema, o resultado foi invertido. O *broker* sem TLS registrou consumo médio de aproximadamente 2,7%, contra 1,85% do *broker* com TLS.

Uma possível explicação para esse comportamento é a distribuição distinta do processamento entre espaço de usuário e espaço de sistema. Enquanto o TLS adiciona operações criptográficas executadas predominantemente em espaço de usuário, o tráfego sem TLS pode demandar maior participação do kernel no gerenciamento das conexões. Em termos absolutos, ambos os valores foram baixos para o hardware utilizado, o que é compatível com a ausência de saturação de recursos observada nos testes de *throughput*.

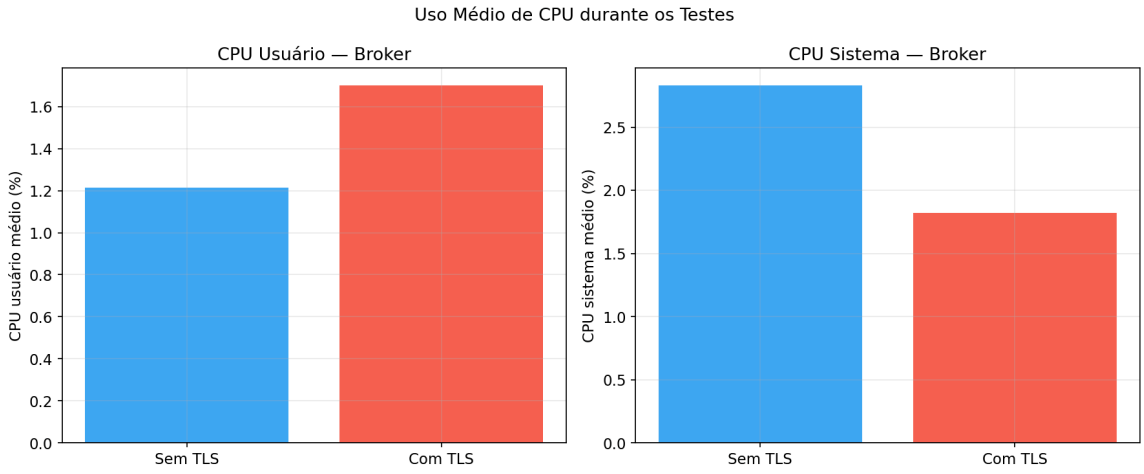

Figura 5.7 – Uso médio de CPU nos *brokers* durante os testes.

A Figura 5.8 apresenta o uso médio de memória. Os dois cenários foram praticamente equivalentes, com aproximadamente 159MB sem TLS e 162MB com TLS, indicando que a manutenção dos contextos TLS por conexão não representou sobrecarga significativa de memória no hardware utilizado. Esse comportamento também foi observado por Gavriilidis et al. (2025), que identificaram que, em hardware mais robusto, o impacto do TLS nos recursos tende a ser absorvido sem comprometer a operação do sistema.

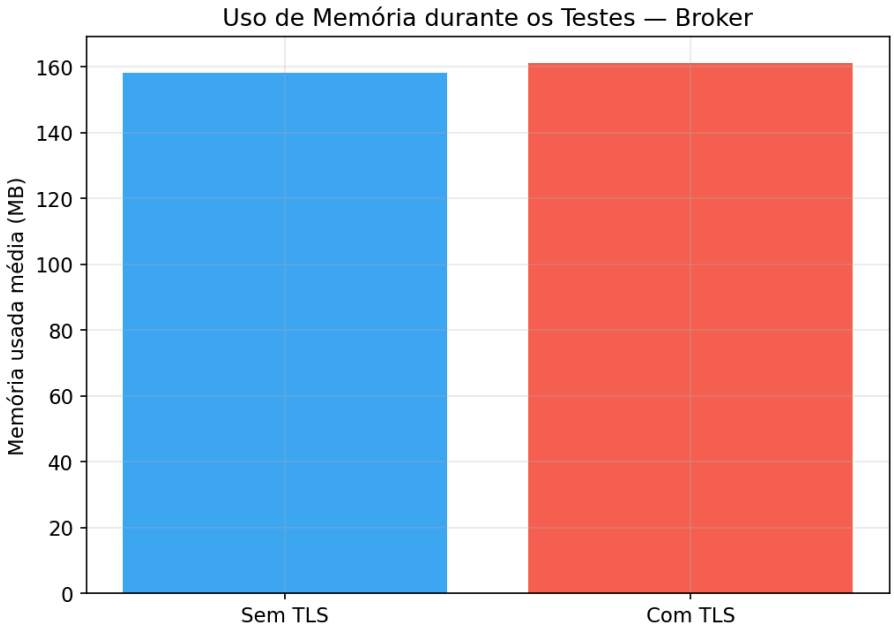

Figura 5.8 – Uso médio de memória nos *brokers* durante os testes.

6. CONCLUSÃO

Este trabalho teve como objetivo avaliar o impacto do uso de TLS no desempenho do *broker* Mosquitto, comparando diferentes cenários de carga em configurações com e sem o protocolo. Com base nos resultados obtidos, é possível responder à pergunta de pesquisa proposta no Capítulo 1: o impacto do TLS é desprezível em cargas moderadas, permanecendo abaixo de 1% até 75 *clients* simultâneos, e se torna mais evidente em cenários de alta carga, quando foram observados sinais de instabilidade a partir de aproximadamente 85 *clients* nas condições testadas.

No *throughput*, o *overhead* do TLS permaneceu praticamente nulo até 75 *clients*, chegando a -9,9% em 100 *clients*. A bateria de saturação mostrou que essa degradação é progressiva, atingindo -12,8% em 110 *clients*, com o *broker* sem TLS mantendo-se estável e previsível em toda a faixa avaliada. Na latência, o TLS apresentou *overhead* de aproximadamente 6 a 7% em volumes menores de mensagens, com essa diferença reduzindo-se e convergindo em volumes maiores. Já no uso de CPU e memória, o *overhead* se mostrou baixo em termos absolutos, com um comportamento interessante na distribuição entre espaço de usuário e espaço de sistema, e consumo de memória praticamente equivalente entre os dois cenários.

Algumas decisões de escopo foram tomadas para manter o experimento viável dentro do tempo disponível. O tamanho de mensagem foi fixado em 256B e o hardware utilizado foi único (e2-medium), evitando uma combinação muito ampla de variáveis (tamanho de mensagem, hardware e carga) que comprometeria a profundidade da análise de cada uma isoladamente. Cabe ressaltar que esse *hardware* é consideravelmente mais robusto do que o utilizado em parte dos trabalhos relacionados discutidos no Capítulo 3, como o Raspberry Pi 4B empregado por Gavriilidis et al. (GAVRIILIDIS; HALKIDIS; PETRIDOU, 2025) e o microcontrolador ESP8266 utilizado como *client* por Prantl et al. (PRANTL et al., 2021). Essa diferença limita a generalização direta dos resultados obtidos para implantações do *broker* em dispositivos mais restritos, como *gateways* de baixo custo ou *hardware* embarcado, cenários nos quais o *overhead* do TLS tende a ser ainda mais pronunciado.

Outra limitação encontrada foi na medição de latência: como o *emqtt-bench* reporta essa métrica apenas de forma agregada e com granularidade insuficiente para as diferenças observadas neste ambiente (conforme discutido na Seção 5.3), foi necessário medir essa métrica separadamente com um único *client*, o que não captura o comportamento de latência sob concorrência real.

Além disso, conforme discutido na Seção 5.2, o monitoramento de CPU e memória foi realizado apenas nas VMs dos *brokers*, não contemplando o *test-runner*

responsável pela geração da carga. Como o estabelecimento e a manutenção de conexões TLS também exigem processamento criptográfico do lado *client*, este trabalho não isola precisamente a contribuição do *broker* Mosquitto e a do próprio gerador de carga para a instabilidade observada a partir de aproximadamente 85 *clients* simultâneos. Trabalhos futuros podem incluir o monitoramento dos recursos do *test-runner*, permitindo isolar com maior precisão o impacto de cada componente nos resultados obtidos.

Como sugestão de trabalho futuro, recomenda-se a variação do tamanho de mensagem, testando valores como 64B, 1KB e 4KB, agora que o comportamento do *broker* por nível de carga já está caracterizado neste trabalho. Outras direções incluem a avaliação de diferentes *cipher suites* (como ChaCha20_Poly1305 e AES_256_GCM) e de versões distintas do TLS, permitindo identificar combinações mais adequadas ao contexto de IoT, bem como a repetição dos testes utilizando *hardware* mais restrito na VM do *broker*, aproximando o ambiente das condições típicas de dispositivos de borda discutidas na literatura relacionada.

Conforme discutido no Capítulo 3, os trabalhos relacionados divergem entre si quanto ao *hardware* utilizado, ao *broker* avaliado e à metodologia de medição empregada, o que dificulta a comparação direta de seus resultados. Diferentemente desses estudos, que em geral avaliam uma única condição de carga, comparam múltiplos *brokers* de forma mais ampla e superficial, ou utilizam *hardware* fortemente restrito, este trabalho isolou um único *broker* (Mosquitto) em um ambiente controlado e reproduzível, provisionado via Terraform e configurado via Ansible, no qual a única variável alterada entre os cenários foi a ativação do TLS. Essa configuração permitiu caracterizar, ao longo de uma ampla faixa de carga (de 1 a 110 *clients*) e a partir de quatro métricas coletadas simultaneamente sob as mesmas condições, o ponto específico em que o *overhead* do TLS se torna relevante, algo não explorado com esse nível de granularidade nos trabalhos analisados.

Assim, este trabalho contribui com evidências quantitativas que podem auxiliar desenvolvedores e engenheiros na decisão sobre o uso de TLS em aplicações MQTT baseadas no Mosquitto, ajudando a preencher a lacuna de padronização identificada nos trabalhos relacionados por meio de uma análise específica, controlada e replicável para esse *broker*.

REFERÊNCIAS BIBLIOGRÁFICAS

Altoros Labs. *A Collection of MQTT Broker Performance Benchmarks 2020–2023*. 2023. Acesso em: 02 nov. 2025. Disponível em: <https://www.altoroslabs.com/blog/a-collection-of-mqtt-broker-performance-benchmarks-2020-2023/>.

Cloudflare. *Do the ChaCha: Better Mobile Performance with Cryptography*. 2019. Acesso em: 02 nov. 2025. Disponível em: <https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/>.

DIZDAREVIC, J.; MICHALKE, M.; JUKAN, A. Engineering and experimentally benchmarking open source MQTT broker implementations. *arXiv*, 2023. Disponível em: <https://arxiv.org/abs/2305.13893>.

EMQX. *Open MQTT Benchmarking Comparison: MQTT Brokers in 2023*. 2023. Acesso em: 02 nov. 2025. Disponível em: <https://www.emqx.com/en/blog/open-mqtt-benchmarking-comparison-mqtt-brokers-in-2023>.

EMQX. *emqtt-bench: Erlang MQTT Benchmark Tool*. 2026. <https://github.com/emqx/emqtt-bench>. Commit ffbef96, branch master. Acesso em: 13 abr. 2026.

GAVRIILIDIS, N. O.; HALKIDIS, S. T.; PETRIDOU, S. Empirical evaluation of TLS-enhanced MQTT on iot devices for v2x use cases. *Applied Sciences*, v. 15, n. 15, p. 8398, 2025. Disponível em: <https://www.mdpi.com/2076-3417/15/15/8398>.

GRIGORIK, I. *High Performance Browser Networking: Transport Layer Security (TLS)*. 2013. Acesso em: 29 out. 2025. Disponível em: <https://hpbn.co/transport-layer-security-tls/>.

HiveMQ. *Cracking MQTT Performance with Automation: Benchmarking and Load Testing*. 2022. Acesso em: 02 nov. 2025. Disponível em: <https://www.hivemq.com/blog/cracking-mqtt-performance-automation-part-2-benchmarking/>.

HiveMQ. *MQTT Security Fundamentals: TLS and Certificates*. 2023. Acesso em: 02 nov. 2025. Disponível em: <https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/>.

LIGHT, R. A. Mosquitto: Server and client implementation of the MQTT protocol. *Journal of Open Source Software*, v. 2, n. 13, p. 265, 2017. Disponível em: <https://joss.theoj.org/papers/10.21105/joss.00265>.

MISHRA, B.; MISHRA, B.; KERTESZ, A. Stress-testing mqtt brokers: A comparative analysis of performance measurements. *Energies*, v. 14, n. 18, 2021. ISSN 1996-1073. Disponível em: <https://www.mdpi.com/1996-1073/14/18/5817>.

OASIS. *MQTT Version 3.1.1 Plus Errata 01*. [S.l.], 2014. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/>.

OASIS. *MQTT Version 5.0 Specification*. [S.l.], 2019. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>.

PRANTL, T. et al. Performance impact analysis of securing MQTT using TLS. 2021. Disponível em: <https://research.spec.org/icpe_proceedings/2021/proceedings/p241.pdf>.

RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. 2018. RFC 8446. Disponível em: <https://datatracker.ietf.org/doc/html/rfc8446>.

SEOANE, V. et al. Performance evaluation of CoAP and MQTT with security support for iot environments. *Computer Networks*, v. 197, p. 108338, 2021. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128621003364>.

SONI, D.; MAKWANA, A. A survey on mqtt: A protocol of internet of things (iot). *International Journal of Computer Applications*, v. 160, n. 7, p. 6–11, 2017. Disponível em: <https://www.researchgate.net/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT>.

THAVAMANI, T.; SINTHUJA, S. MQTT messages – an overview. *International Journal of Multidisciplinary and Current Research*, 2021. Disponível em: <https://ijmcr.in/index.php/ijmcr/article/view/311>.