UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Henrique Bueno de Morais

Impacto da computação de borda na qualidade de serviço de redes em cenários MMTC

Blumenau
2025

Henrique Bueno de Morais

Impacto da computação de borda na qualidade de serviço de redes em cenários MMTC

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Dr. Ciro André Pitz

Blumenau
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Morais, Henrique Bueno de
Impacto da computação de borda na qualidade de serviço de redes em cenários MMTC / Henrique Bueno de Morais ; orientador, Ciro André Pitz, 2025.
52 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2025.

Inclui referências.

1. Engenharia de Controle e Automação. 2. computador de borda. 3. redes de comunicação. 4. QoS. 5. MMTC. I. Pitz, Ciro André. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Henrique Bueno de Morais

Impacto da computação de borda na qualidade de serviço de redes em cenários MMTC

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 15 de dezembro de 2025.

Banca Examinadora:

Documento assinado digitalmente
Ciro Andre Pitz
Data: 18/12/2025 19:55:03-0300
CPF: ***.728.089-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Ciro André Pitz
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Carlos Roberto Moratelli
Data: 18/12/2025 19:42:02-0300
CPF: ***.999.369-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Carlos Roberto Moratelli
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Adao Boava
Data: 18/12/2025 19:57:07-0300
CPF: ***.640.969-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Adão Boava
Universidade Federal de Santa Catarina

Dedico este trabalho a minha família, amigos, e a todos que se interessarem.

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais Marcelo e Ligia, que foram e sempre serão as minhas maiores inspirações, seu apoio foi fundamental para o meu desenvolvimento profissional e pessoal. Agradeço também ao meu irmão Lucas, por me incentivar a entrar no curso de Engenharia de Controle e Automação, e pelo suporte durante toda a minha jornada acadêmica.

Agradeço a todos os meus amigos e colegas que me acompanharam nesta caminhada e tornaram essa aventura em algo inesquecível. Vocês me proporcionaram risadas, companhia e apoio, fundamentais para que essa jornada foi possível.

Agradeço também ao Prof. Dr. Carlos Roberto Moratelli, pelo apoio durante toda a minha graduação com conselhos sempre pertinentes e empréstimos de equipamentos durante o curso.

Por fim, meus agradecimentos também a Universidade Federal de Santa Catarina por proporcionar uma formação sólida e de qualidade, e por transformar a forma como vejo o mundo. Em especial, agradeço ao meu orientador Prof. Dr. Ciro André Pitz, pela colaboração e paciência, que foram cruciais para o sucesso deste trabalho.

RESUMO

Este trabalho analisa a influência da computação de borda na qualidade de serviço (QoS) de redes em cenário MMTC (Massive Machine-Type Communications). O estudo estabelece como objetivo principal avaliar como a descentralização do processamento pode reduzir a latência, estabilizar o tráfego e melhorar o desempenho de sistemas com grande número de dispositivos conectados. Para isso, o trabalho implementa um ambiente experimental composto por sensores simulados em *containers Docker*, um nó de borda físico e um servidor em nuvem, permitindo a criação de cenários que representam diferentes condições de carga e congestionamento. A análise dos cenários permite observar o comportamento da rede com e sem o uso da computação de borda, destacando a comparação entre métricas como latência, *jitter*, *throughput* e tempo de resposta ao ambiente. Os resultados mostram que a utilização de um nó de borda reduz significativamente o atraso fim-a-fim, melhora a estabilidade temporal das transmissões e aumenta a resiliência da rede em condições adversas. O processamento local diminui a sobrecarga sobre o servidor em nuvem, evitando o acúmulo de pacotes e reduzindo o risco de congestionamento em redes densas. Além disso, os experimentos indicam que o nó de borda reduz o tempo de resposta ao meio, fornecendo um comportamento mais eficiente em aplicações que exigem reação imediata. A discussão evidencia que a computação de borda se apresenta como uma solução eficaz para cenários MMTC, oferecendo benefícios em escalabilidade, desempenho, confiabilidade e utilização dos recursos.

Palavras-chave: computador de borda; redes de comunicação; QoS; MMTC

ABSTRACT

This work analyzes the influence of edge computing on the quality of service (QoS) in MMTC (Massive Machine-Type Communications). The main objective is to evaluate how decentralized processing can reduce latency, stabilize traffic, and improve the performance of systems composed of a large number of connected devices. To achieve this goal, the study implements an experimental environment composed of simulated sensors, an edge node, and a cloud server, enabling the creation of scenarios that represent different levels of load and network congestion. The analysis of the scenarios makes it possible to observe network behavior with and without edge computing, highlighting key metrics such as latency, jitter, throughput, and environmental response time. The results show that the use of an edge node significantly reduces end-to-end delay, improves temporal stability of transmissions, and increases network resilience under adverse conditions. Local processing decreases the load on the cloud server, preventing packet accumulation and reducing the risk of congestion in dense networks. Additionally, the experiments indicate that the edge node reduces the round-trip time, providing more efficient behavior for applications requiring immediate reaction. The discussion demonstrates that edge computing is an effective solution for MMTC scenarios, offering benefits in scalability, performance, reliability, and resource utilization. The study concludes that the integration between edge and cloud contributes to a more efficient and intelligent network architecture, reinforcing its role as an essential strategy for future generations of distributed communication systems.

Keywords: edge computing; telecommunication networks; QoS; MMTC.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Access Point Interface*
AWS	*Amazon Web Services*
EC2	*Amazon Elastic Compute Cloud*
ETSI	*European Telecommunications Standard Institute*
HTB	*Hierarchical Token Bucket*
IoT	*Internet of Things*
IP	*Internet Address*
LSTM	*Long short-term memory*
MMTC	Massive Machine-Type Communications
MQTT	*Message Queuing Telemetry Transport*
QoS	Qualidade de serviço
SSH	*Secure Shell*
TC	*Traffic Control*
vCPU	*Virtual Central Process Unit*

SUMÁRIO

1 INTRODUÇÃO

Com o constante aumento de dispositivos inteligentes nos mais diversos setores da indústria, aliado a processamentos de dados cada vez mais complexos, a necessidade de qualidade de serviço tem se tornado um fator crítico em redes de telecomunicações. Segundo pesquisa realizada pela IoT Analytics em 2024, existem aproximadamente 18,8 bilhões de dispositivos inteligentes em funcionamento no mundo, com estimativas de atingir até 41,1 bilhões até 2030 (IOT ANALYTICS GMBH, 2024). Neste cenário, torna-se inviável a utilização de uma topologia de rede monolítica, uma vez que a qualidade de serviço, a saber: a latência, eficiência espectral, *jitter*, dentre outras métricas, tendem a piorar rapidamente com o aumento da quantidade de dispositivos geradores de dados (AI; PENG; ZHANG, 2018)

Sob esse prisma, a computação de borda, aliada à computação em nuvem, se consolidou como uma aplicação necessária na viabilização do processamento e utilização da grande quantidade de dados gerados. A computação de borda, como definição proposta pelo ETSI (*European Telecommunications Standard Institute*), compreende um modelo de arquitetura descentralizada que estende recursos de nuvem para regiões de borda da rede (ETSI, 2023). Dessa forma, pode-se realizar os processos críticos no que tange aos recursos computacionais e tempo de resposta nos nós de borda, mais próximos fisicamente ao meio gerador de dados, exportando apenas dados relevantes para o *core* da rede. Essa abordagem garante melhor qualidade de serviço, reduz a latência e melhora a eficiência espectral e energética do sistema.

Nesse contexto, este trabalho tem como objetivo avaliar a qualidade de serviço em redes do tipo MMTC (Massive Machine-Type Communications), com a implementação de um nó de borda e um servidor em nuvem. Ao final do projeto, espera-se comprovar a relevância da computação de borda na mitigação de problemas de latência, sobrecarga de banda e degradação do desempenho da rede, bem como demonstrar que sua aplicação contribui para maior escalabilidade e confiabilidade em redes MMTC.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal analisar, comparativamente, os efeitos da computação de borda na melhoria da qualidade de serviço de redes MMTC sob diferentes condições e ambientes.

1.1.2 Objetivos Específicos

A seguir são listados os objetivos específicos deste trabalho:

- Implementar um ambiente de simulação de cenários de rede MMTC.

- Implementar um nó de borda e um servidor em nuvem, de modo a possibilitar a simulação de diferentes cenários em redes MMTC.
- Configurar algoritmos de processamento de sinais tipicamente utilizados em redes MMTC.
- Avaliar comparativamente a qualidade de serviço e métricas de rede com e sem a utilização do nó de borda.

1.1.3 Estrutura do documento

O Capítulo contempla os objetivos gerais e específicos deste trabalho. O Capítulo 2 contém a revisão de literatura de informações referentes ao tema, pertinentes para a compreensão dos tópicos aprofundados neste trabalho. Os procedimentos e ferramentas utilizadas estão descritas no Capítulo 3, enquanto o Capítulo 4 apresenta os resultados obtidos através de experimentos. Por fim, no Capítulo 5 são apresentadas as conclusões finais e sugestões de pesquisas sobre o tema.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos essenciais para o desenvolvimento do trabalho, abordando a computação de borda e sua importância em cenários MMTC, os mecanismos de QoS, a integração com servidores em nuvem, além de algoritmos de predição e ferramentas utilizadas na implementação dos experimentos.

2.1 COMPUTAÇÃO DE BORDA

A computação de borda é um paradigma computacional distribuído que desloca parte do processamento e armazenamento de dados para nós localizados próximos à origem da informação, como sensores, atuadores e dispositivos de IoT (*Internet of Things*). Diferentemente da computação em nuvem tradicional, em que todo o processamento ocorre em *datacenters* centralizados, a computação de borda reduz a latência, melhora a eficiência espectral e aumenta a escalabilidade da rede (MAO *et al.*, 2017; SHI *et al.*, 2016) ao aproximar o processamento do meio gerador de dados.

A computação de borda se tornou fundamental no contexto das redes 5G e aplicações de IoT massiva. Em particular, no cenário de MMTC, previsto como um dos pilares do 5G, milhões de dispositivos geram tráfego simultâneo, exigindo soluções de escalabilidade e redução de sobrecarga na rede (ATZORI; IERA; MORABITO, 2010; BENNIS; DEBBAH; POOR, 2018). Nessa perspectiva, a computação de borda contribui para assegurar a qualidade de serviço ao realizar filtragem, pré-processamento e análise local dos dados antes de encaminhá-los à nuvem (ZHOU, Z. *et al.*, 2019).

2.1.1 Arquitetura da Computação de Borda

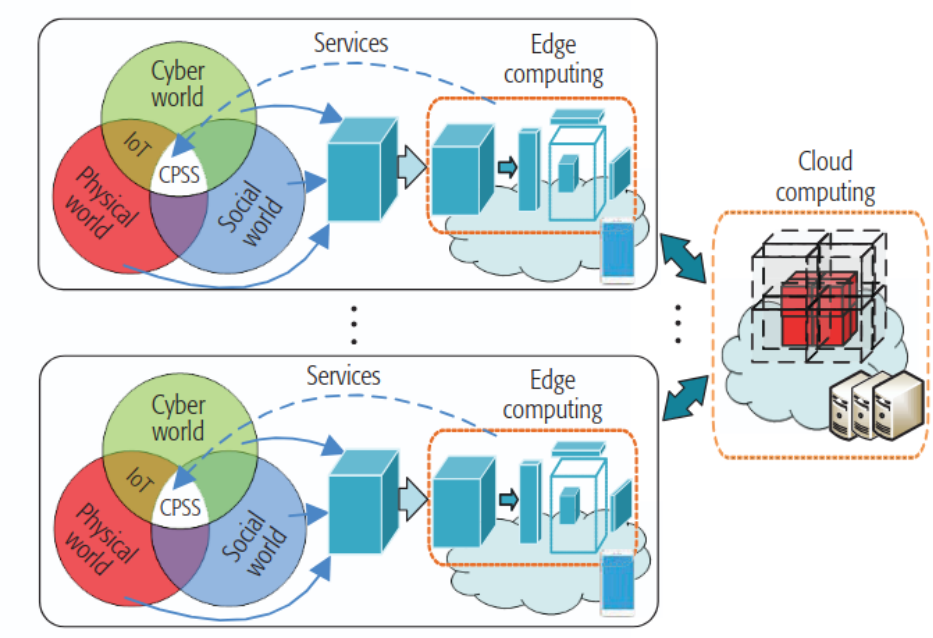
A arquitetura típica da computação de borda pode ser dividida em três camadas principais:

- Dispositivos de borda: sensores, atuadores, dispositivos móveis e equipamentos IoT responsáveis pela coleta de dados.
- Nós de Borda: servidores locais, *gateways* ou micro *datacenters* capazes de realizar processamento e armazenamento intermediário.
- Nuvem: *datacenters* centralizados responsáveis por análises complexas, persistência de dados em longo prazo e coordenação global do sistema.

Esse modelo hierárquico permite reduzir a quantidade de dados transmitidos à nuvem, mitigando congestionamentos e diminuindo a latência de resposta (SHI *et al.*, 2016). A Figura 1 ilustra as interfaces de comunicação entre o ambiente físico e o servidor em nuvem, interligados por um nó de borda, identificado na figura como *Edge Computing*. Nesse contexto, os dados brutos são processados localmente pelo computador de borda, que estabelece uma comunicação bidirecional com o servidor em nuvem. Essa abordagem

proporciona maior controle sobre a rede, reduz a sobrecarga de dados na nuvem e melhora a eficiência da rede de forma geral.

Figura 1 – Diagrama de aplicações de computação de borda.



Fonte: (ATZORI; IERA; MORABITO, 2010)

2.1.2 Computação de Borda em MMTC

O papel da computação de borda é ainda mais relevante nas redes MMTC, em que a QoS (Qualidade de serviço) precisa ser garantida mesmo diante da elevada densidade de dispositivos conectados. Estratégias de processamento local e de agregação de dados contribuem para otimizar o uso da rede, reduzir a sobrecarga no núcleo e aumentar a eficiência energética dos dispositivos finais (BENNIS; DEBBAH; POOR, 2018; ZHOU, Z. *et al.*, 2019).

Dessa forma, a computação de borda se consolida como uma tecnologia-chave para atender aos requisitos de escalabilidade, confiabilidade e desempenho das futuras redes MMTC.

2.2 QoS EM COMPUTAÇÃO DE BORDA

QoS em ambientes de borda refere-se ao conjunto de mecanismos e políticas que asseguram níveis aceitáveis de desempenho (latência, vazão, disponibilidade, confiabilidade e, quando relevante, segurança e privacidade) para aplicações com requisitos heterogêneos. A descentralização, heterogeneidade de recursos (CPU, memória, conectividade) e a mo-

bilidade dos dispositivos no meio tornam a garantia de QoS um desafio distinto daquele enfrentado em nuvem tradicional.

Diversos estudos recentes têm investigado estratégias para otimizar a QoS na computação de borda. Em (MUTICHIRO; TRAN; KIM, 2021) foram propostas políticas de escalonamento orientadas a tempo de serviço para *clusters* IoT/*edge*, mostrando ganhos expressivos no atendimento de prazos (*deadlines*) e melhor utilização dos recursos em cenários com requisitos temporais críticos. Já em (ZHOU, J. *et al.*, 2024) foi explorado o uso de aprendizado federado em arquiteturas híbridas entre nó de borda e nuvem, demonstrando que modelos distribuídos podem prever congestionamentos e ajustar dinamicamente a alocação de recursos, reduzindo significativamente a latência percebida em dispositivos de IoT inteligente. Nesse mesmo sentido, em (CAO *et al.*, 2024) foi realizada uma revisão sistemática sobre otimização de custos em ambientes de borda, discutindo como estratégias de posicionamento e escalonamento dos nós impactam diretamente a QoS. Os autores apontam, ainda, para a ausência de avaliações empíricas realistas, o que evidencia uma lacuna importante a ser explorada por trabalhos experimentais.

Outro aspecto em crescente destaque é a interdependência entre segurança, energia e QoS. Em (MARWAN *et al.*, 2024) foi proposta uma otimização conjunta desses parâmetros em sistemas *edge–cloud*, enfatizando que medidas adicionais de segurança podem degradar a QoS e, portanto, devem ser co-otimizadas. Em uma perspectiva mais ampla, em (ERGEN *et al.*, 2024) foi apresentada uma avaliação abrangente das arquiteturas e *frameworks* de computação de borda, destacando implicações arquiteturais para QoS e indicadores práticos para mensuração em redes sem fio, particularmente em cenários 5G.

A análise desses trabalhos evidencia a relevância do estudo de QoS em ambientes de borda. Os parâmetros mais comumente avaliados incluem latência fim-a-fim, *jitter*, *throughput*, perda de pacotes, tempo de resposta de serviços e utilização de recursos. Em termos metodológicos, observa-se a utilização de medições experimentais em ambientes reais, simulações para avaliação de escalabilidade, algoritmos de orquestração inteligente baseados em heurísticas ou aprendizado de máquina, e estudos de *trade-offs* entre segurança, custo e desempenho (ZHOU, J. *et al.*, 2024; MARWAN *et al.*, 2024).

Diante desse panorama, percebe-se que, embora haja avanços significativos em algoritmos de alocação e orquestração, ainda existe carência de avaliações empíricas integradas que combinem captura de tráfego, métricas de aplicação e medições de recursos em cenários reais de borda voltados a MMTC.

Nas seções 2.3, 2.4 e **??** deste capítulo são apresentadas as principais ferramentas, serviços e conceitos fundamentais para a análise de QoS em cenários de computação de borda, incluindo métodos de captura de tráfego, plataformas de orquestração de serviços e métricas de avaliação.

2.3 SERVIDOR EM NUVEM AWS

A AWS (*Amazon Web Services*) é uma plataforma de computação em nuvem amplamente utilizada, que oferece serviços sob demanda para computação, armazenamento, banco de dados e redes, entre outros (AMAZON WEB SERVICES, INC., 2024). Diferentemente de servidores locais, a AWS permite o provisionamento elástico de recursos, ajustando a capacidade conforme a demanda e adotando o modelo de pagamento conforme o uso, conhecido como *pay-as-you-go*, o que reduz custos e aumenta a flexibilidade operacional. Neste sentido, o serviço EC2 (*Amazon Elastic Compute Cloud*) é o principal responsável pela criação e gerenciamento de instâncias virtuais, que funcionam como servidores configuráveis em diferentes níveis de processamento, memória e armazenamento.

No contexto deste trabalho, o uso da AWS possibilita a centralização do processamento e armazenamento de dados em um ambiente distante de onde se encontram os sensores geradores de dados, permitindo uma representação real de um ambiente de computação em nuvem.

2.4 ALGORITMOS DE PREDIÇÃO BASEADOS EM LSTM

Entre os métodos de predição aplicados em sistemas de processamento distribuído e em nós de borda, destacam-se as redes neurais recorrentes do tipo LSTM (*Long short-term memory*). Propostas em (HOCHREITER; SCHMIDHUBER, 1997), as redes LSTM foram desenvolvidas para superar as limitações das redes recorrentes convencionais, a saber: a extrapolação das entradas durante o treinamento. Sua estrutura interna é composta por portas de entrada, esquecimento e saída, que controlam o fluxo de informação ao longo do tempo, permitindo à rede armazenar e recuperar dependências de longo prazo em séries temporais.

Em aplicações de predição de dados sensoriais e séries temporais não estacionárias, as LSTM têm se mostrado eficazes por sua capacidade de modelar relações temporais complexas e dinâmicas. Conforme demonstrado em (MALHOTRA *et al.*, 2016), modelos baseados em LSTM, são capazes de reconstruir comportamentos normais em sinais multivariados e detectar anomalias com base no erro de reconstrução, mesmo em cenários com ruído, periodicidade variável ou baixa previsibilidade.

Evoluções recentes, como o modelo *LSTM-Attention-LSTM* proposto em (WEN; LI, 2023), incorporam mecanismos de atenção entre o codificador e o decodificador, permitindo que a rede atribua pesos diferenciados às entradas mais relevantes no tempo. Essa modificação melhora a exatidão em predições com janelas temporais longas e reduz a perda de informação em sequências extensas, tornando o modelo mais robusto e adequado a aplicações em contextos de IoT, previsão de séries multivariadas e análise de sinais em borda. Dessa forma, a escolha do algoritmo LSTM neste trabalho justifica-se pela sua capacidade de capturar padrões temporais de longo alcance, lidar com dados não lineares e

operar de forma eficiente em sistemas com restrições computacionais, mantendo exatidão satisfatória em tarefas de predição e detecção em tempo real.

2.5 PRINCIPAIS FERRAMENTAS

2.5.1 *Docker*

O Docker é uma plataforma para criação, distribuição e execução de aplicações em ambientes isolados chamados *containers* (DOCKER, 2025). Diferentemente de máquinas virtuais completas, os *containers* compartilham o mesmo *kernel* do *host*, mas fornecem isolamento por meio de espaços de trabalho diferentes, denominados *namespaces* e limitação de recursos, o que permite empacotar aplicações e suas dependências em unidades leves, portáteis e facilmente reproduzíveis.

O Docker utiliza imagens, criadas a partir de um arquivo *Dockerfile*, para definir o ambiente e as instruções de execução. A partir dessas imagens, são iniciados os *containers*, que funcionam de forma independente e podem ser facilmente gerenciados pela central de gerenciamento do Docker. Conforme ilustrado na Figura 2, a aplicação geral do Docker é executada em uma máquina principal, denominada *host*. Essa máquina é responsável por executar e gerenciar aplicações isoladas, chamadas *apps*, que possuem suas próprias bibliotecas e dependências específicas.

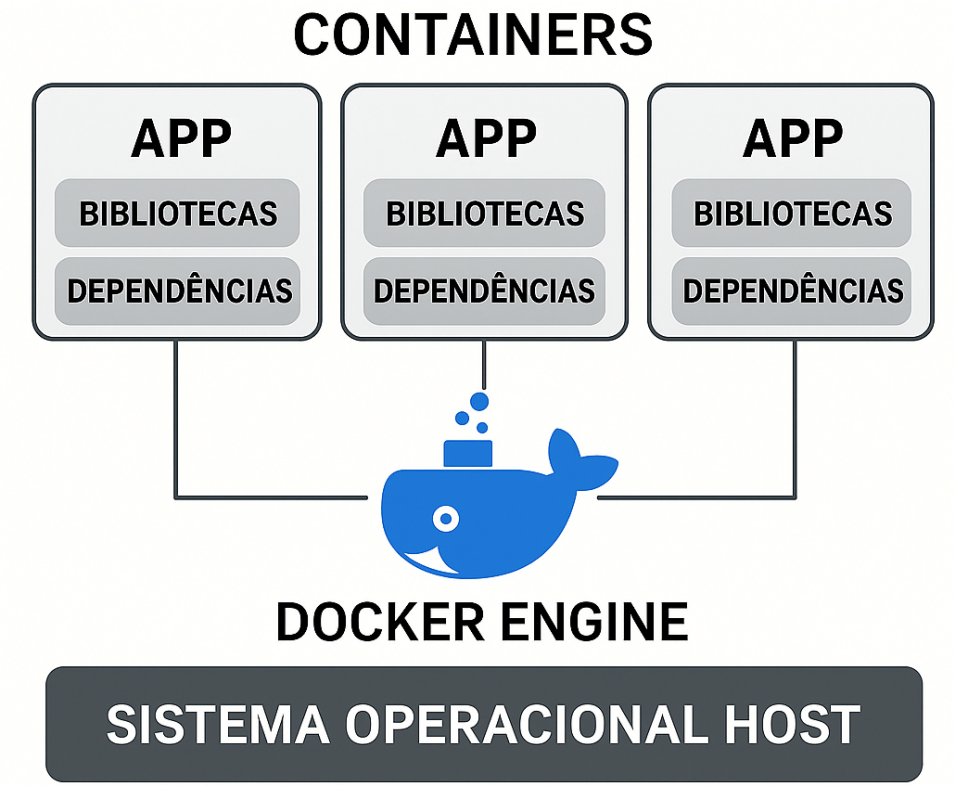
A plataforma também fornece recursos para redes, armazenamento e segurança, permitindo a criação de ambientes complexos compostos por múltiplos serviços. Graças à sua portabilidade e facilidade de uso, o Docker é amplamente utilizado em desenvolvimento, testes e implantação de aplicações. Neste trabalho, o Docker será utilizado para simular os ambientes MMTC, onde cada *container* atuará como um *cluster*, ou conjunto, de dezenas ou centenas de sensores, gerando dados simultaneamente para a rede.

2.5.2 *Wireshark*

O *Wireshark* é considerado a ferramenta de captura e análise de tráfego de rede mais popular na atualidade, em grande parte devido à sua interface gráfica intuitiva e às suas poderosas opções de filtragem e análise de pacotes, aliado ao fato de ser totalmente *open source* (GOYAL, P.; GOYAL, A., 2017).

Uma de suas principais características é a capacidade de operar tanto em modo promíscuo quanto em modo monitor, permitindo ao usuário capturar todo o tráfego que passa pela interface de rede, mesmo aquele que não é destinado diretamente à máquina que realiza a captura. Além disso, o *software* organiza automaticamente os pacotes por protocolos, utilizando esquemas de cores que facilitam a interpretação, e oferece filtros avançados que possibilitam isolar fluxos de interesse em tempo real (HASHIM *et al.*, 2023), conforme pode ser visualizado na Figura 3. Tais funcionalidades tornam o *Wireshark* uma

Figura 2 – Diagrama da arquitetura do Docker.



Fonte: Do Autor (2025).

ferramenta amplamente utilizada em ensino, pesquisa e também em atividades de auditoria, segurança e análise forense de redes (NDATINYA *et al.*, 2015).

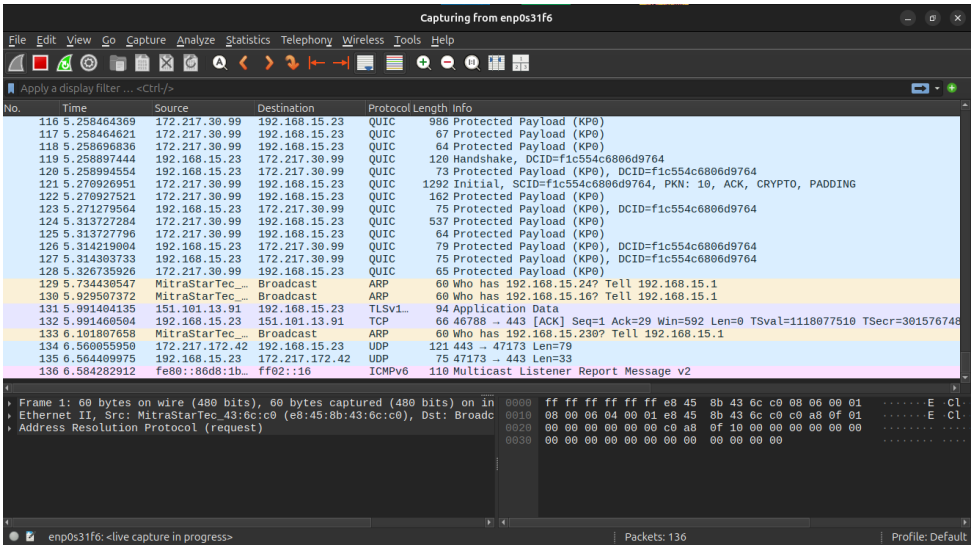
Em cenários MMTC, essa ferramenta se torna muito útil para realizar capturas de pacotes e analisar a largura da banda utilizada, taxa de pacotes que foram, ou não, recebidos pelo receptor, dentre outras informações relevantes no estudo presente neste trabalho.

2.5.3 *Traffic Control*

O TC (*Traffic Control*) é o subsistema do *kernel* Linux responsável pelo gerenciamento e controle do tráfego de rede em interfaces de comunicação. Ele provê mecanismos para manipulação de pacotes em diferentes níveis, permitindo a implementação de políticas de QoS, modelagem de tráfego, priorização, limitação de banda, simulação de condições de rede e até filtragem avançada (CONTROL, 2018).

O TC também desempenha um papel fundamental na simulação e controle de condições de rede, sendo amplamente utilizado em ambientes de teste, desenvolvimento e pesquisa. Através da combinação de parâmetros, denominados *qdiscs*, classes e filtros, é possível modelar parâmetros como latência, *jitter*, perda de pacotes e limitação da banda utilizada. Uma das ferramentas mais utilizadas para essa finalidade é o módulo `netem`

Figura 3 – Interface gráfica do Wireshark para captura de pacotes em tempo real.



Fonte: Do Autor (2025).

(*Network Emulator*), integrado ao TC. O `netem` permite a introdução de atrasos artificiais em pacotes, com distribuição determinística ou probabilística, de modo a representar diferentes cenários de comunicação.

Além disso, o `netem` possibilita a injeção controlada de perda de pacotes, tanto de forma aleatória quanto correlacionada, reproduzindo situações típicas de congestionamento ou instabilidade de enlace. Também é possível simular duplicações, reordenação de pacotes e corrupção de dados, fornecendo um ambiente de teste realista para aplicações sensíveis a variações de rede, como sistemas de controle remoto, aplicações multimídia e protocolos de comunicação industrial.

Outro módulo muito utilizado para simulação de cenários de rede é o módulo HTB (*Hierarchical Token Bucket*). Este atua de modo a controlar a largura de banda disponível, permitindo a criação de classes hierárquicas que definem limites mínimos e máximos de taxa para diferentes fluxos de tráfego. Com isso, é possível priorizar determinados tipos de pacotes ou aplicações. Neste trabalho, esta ferramenta será utilizada para emular ambientes adversos na rede MMTC.

2.5.4 MQTT

O protocolo MQTT (*Message Queuing Telemetry Transport*) é amplamente utilizado em aplicações de IoT devido à sua natureza leve, confiável e de fácil implementação. Desenvolvido originalmente pela IBM em 1999, o MQTT segue o paradigma de comunicação *publish/subscribe*, no qual um intermediário, denominado *broker*, gerencia a troca de mensagens entre os dispositivos produtores e consumidores de dados (LIMA *et al.*, 2025). Essa arquitetura desacoplada reduz a complexidade dos nós terminais e facilita a

escalabilidade da rede, sendo particularmente vantajosa em ambientes com grande número de dispositivos heterogêneos, como ocorre em redes MMTC.

Um dos principais motivos que tornam o MQTT adequado para sistemas IoT é sua baixa sobrecarga de comunicação. O protocolo utiliza cabeçalhos reduzidos e mantém conexões persistentes, o que minimiza o consumo de largura de banda e energia — fatores críticos em dispositivos embarcados e sensores alimentados por bateria (HMISSI; OUNI, 2022). Além disso, o MQTT oferece três níveis de QoS, permitindo que o desenvolvedor equilibre entre confiabilidade e latência de acordo com os requisitos da aplicação. Essa flexibilidade o torna aplicável tanto a cenários de monitoramento contínuo, que toleram pequenas perdas de pacotes, quanto a sistemas críticos, que exigem entrega garantida de mensagens.

Outro diferencial importante é sua robustez em condições de conectividade instável, característica comum em ambientes IoT distribuídos. Recursos como mensagens retidas, sessões persistentes e o mecanismo de *Last Will and Testament* (LWT) permitem que o sistema mantenha a consistência mesmo quando há falhas temporárias de comunicação (HANIF; ILYAS, 2024). Dessa forma, o MQTT se destaca em aplicações de monitoramento remoto, automação residencial e industrial, e agricultura inteligente, onde há intermitência de rede e dispositivos frequentemente entram em modo de economia de energia.

Por fim, estudos recentes apontam que arquiteturas baseadas em MQTT distribuído, como o TD-MQTT (*Transparent Distributed* MQTT), podem melhorar a escalabilidade e reduzir a latência em sistemas com alto volume de dispositivos e mensagens (HMISSI; OUNI, 2022). Assim, o protocolo se consolida como uma solução eficaz e adaptável para comunicação em redes IoT e MMTC, conciliando simplicidade de implementação, eficiência energética e confiabilidade na transmissão de dados.

2.5.5 Grafana

Para a visualização das *features*, optou-se por utilizar a ferramenta Grafana Server em ambiente de nuvem. O Grafana é uma aplicação de código aberto voltada para a observabilidade e visualização de dados, permitindo a criação de dashboards interativos e altamente personalizáveis (GRAFANA LABS, 2025). A ferramenta suporta diversas fontes de dados, sendo amplamente utilizada por equipes de desenvolvimento e análise para o monitoramento de sistemas, aplicações e infraestrutura em tempo real.

Além disso, o Grafana oferece recursos avançados como integração com alertas, transformações de dados e suporte a *plugins* que estendem suas funcionalidades. Está disponível tanto em versões auto-hospedadas quanto como serviço gerenciado na nuvem, conhecido como Grafana Cloud (GRAFANA LABS, 2025). Para a coleta de dados, o Grafana se conecta diretamente à porta de comunicação de um banco de dados, realizando extrações por meio de *queries* específicas, adaptadas ao tipo de banco utilizado. Essa abordagem facilita a implementação de painéis de visualização com taxas de atualização

ajustáveis conforme a necessidade da aplicação.

Neste trabalho, o Grafana será utilizado em conjunto com um banco de dados InfluxDB, com o objetivo de disponibilizar *features* relacionadas aos sinais provenientes do meio e à qualidade da transmissão.A escolha do *InfluxDB* se justifica por sua simplicidade de implementação, uma vez que oferece APIs HTTP simples para escrita e leitura, por sua alta velocidade de operação e por seu amplo uso de mercado, especialmente em aplicações de séries temporais, monitoramento e IoT (INFLUXDATA, 2025). Além disso, o InfluxDB apresenta excelente desempenho na ingestão de dados em tempo real, compressão eficiente de armazenamento e escalabilidade para lidar com grandes volumes de pontos por segundo. Essa integração permitirá o acompanhamento contínuo dos dados, contribuindo para a análise e interpretação dos fenômenos observados.

3 ARQUITETURA E CENÁRIOS PROPOSTOS

Nesta seção estão descritos a arquitetura, as configurações de ambiente e os procedimentos técnicos que viabilizam a simulação, coleta e análise dos dados nos cenários propostos. O processo de desenvolvimento foi dividido em três partes, a saber: a configuração do servidor em nuvem na plataforma AWS; a configuração e implementação do nó de borda em um RaspberryPi; e a implementação propriamente dita dos cenários simulados.

3.1 CRIAÇÃO DOS *CONTAINERS*

Para a criação dos *containers* resposnsáveis pela geração dos dados, faz-se necessário a criação de um ambiente *Docker*, composto de dois arquivos principais: o *docker-compose*, e o *Dockerfile*. O primeiro arquivo tem como objetivo estruturar o ambiente de execução, onde são informadas as variáveis de ambiente e o nome da cada *container* para posterior gerenciamento. Assim, estruturou-se o *docker-compose* de forma modular, possibilitando a replicação de *containers*, cada um responsável por um conjunto n de sensores simulados, conforme pode ser visualizado no Código 3.1.

Neste arquivo, estão contidos todos os ambientes que serão instanciados pela ferramenta *Docker*, juntamente com todas as variáveis de ambiente que serão utilizados no algoritmo executado. Neste caso, contém-se o identificador do primeiro sensor da instância, o número de sensores que serão criados, o IP do *broker* MQTT, bem como a sua porta de comunicação, o tópico de inscrição do MQTT e o intervalo, em segundos, de geração dos valores simulados.

Código 3.1 – Estrutura do arquivo `docker-compose.yml`

```
version: '3.8'
services:
sensor_group_1:
build: .
environment:
- START_SENSOR_ID=1
- NUM_SENSORS=1
- MQTT_BROKER=34.227.100.182
- MQTT_PORT=1883
- MQTT_TOPIC=sensores/termopar
- INTERVAL=2
network_mode: host

sensor_group_2:
build: .
environment:
- START_SENSOR_ID=2
- NUM_SENSORS=1
- MQTT_BROKER=34.227.100.182
```

```
20    - MQTT_PORT=1883
21    - MQTT_TOPIC=sensores/termopar
22    - INTERVAL=2
23    network_mode: host
```

Da mesma forma, o *Dockerfile* tem o papel de ditar os comandos que serão executados no ambiente, bem como o algoritmo que será executado. Além disso, faz-se necessário definir no arquivo a linguagem de programação utilizada e a sua versão, neste caso *Python* v3.12.3, para emular em cada um dos *containers*. O arquivo *Dockerfile* utilzado, neste caso, pode ser visualizado nO Código 3.2.

Código 3.2 – Estrutura do Dockerfile.

```
1  FROM python:3.11-slim
2  WORKDIR /app
3  COPY simulador_meio_nuvem.py .
4  RUN pip install paho-mqtt
5  CMD ["python", "simulador_meio_nuvem.py"]
```

Por fim, tendo configurado ambos os arquivos, pode-se então dar início a construção e execução dos *containers*. Para isso, utiliza-se o comando apresentado no Código 3.3.

Código 3.3 – Comando para criação e inicialização dos *containers*.

```
1  $  sudo docker compose build && sudo docker compose up
```

3.1.1 Geração dos dados simulados de sensores

Em cada *container* Docker é executada a mesma aplicação, que consiste em um simples mecanismo de geração de valores pseudo-aleatórios seguindo uma distribuição uniforme com valores entre 20 e 30 do tipo ponto flutuante com precisão de duas casas decimais. Os valores foram definidos de forma arbitŕaria, seguindo uma amplitude em conformidade com um sensor de temperatura. Uma nova mensagem é gerada a cada 2 segundos para cada sensor em cada *container*, de modo que para um número x de sensores simulados, serão geradas $0.5x$ mensagens a cada segundo.

O pacote de dados transmitidos por cada um deles é composto de três elementos apenas, a saber: um identificador do sensor, denominador $sensor_{id}$, o valor do sinal gerado, denominado de *temperatura*, e o *timestamp* antes da transmissão da mensagem. O *timestamp* gerado está no formato *Unix*, isto é, o número de segundos que se passaram desde a data conhecida como *Epoch Date*, definida como sendo dia 1 de janeiro de 1970. No Código 3.4 pode-se observar, em formato JSON, um exemplo de pacote de dados gerado.

Código 3.4 – Conteúdo de um pacote de dados transmitidos pelo meio.

```
1  {
2    "sensor_id": "sensor_724",
3    "temperatura": 20.03,
```

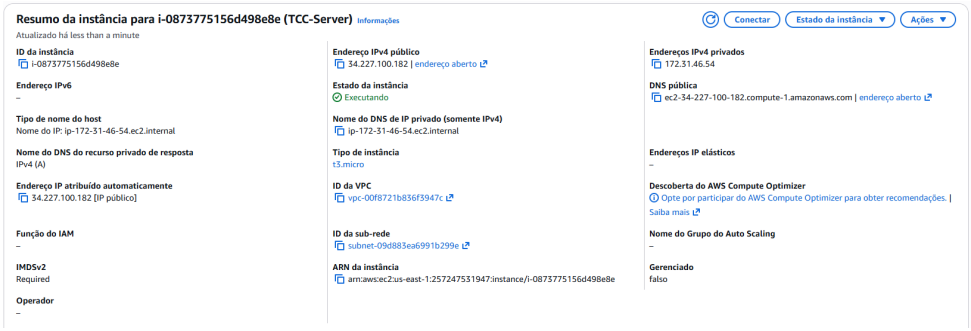
```
    "timestamp": 1762404006.1573186
}
```

3.2 CRIAÇÃO DO SERVIDOR AWS

Para criar o servidor AWS, fez-se necessário realizar alguns procedimentos e configurações de ambiente e de rede. Inicialmente, utilizando-se uma conta credenciada no gerenciador de servidores AWS, pode-se escolher dentre uma série de aplicações modulares para compor o servidor. Neste trabalho, no entanto, fez-se necessário apenas a criação de um serviço do tipo EC2.

Criando uma instância EC2 gratuita, com 2 CPUs virtuais (vCPU) e 1 Gb de memória RAM e Ubuntu *headless* v22.04 como sistema operacional. A partir disso, será providenciado um IP público para acesso, bem como informações relevantes sobre o sistema, conforme pode ser visualizado na Figura 4.

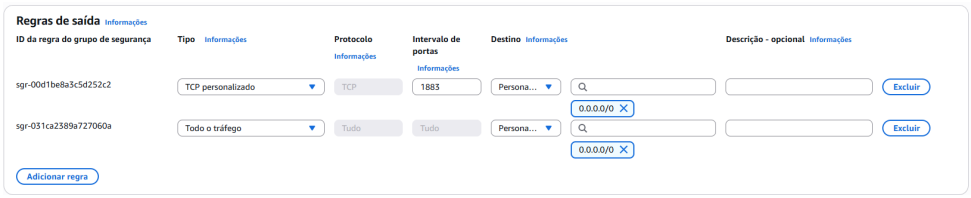
Figura 4 – Informações referentes a instância EC2 criada no servidor AWS.



Fonte: Do Autor (2025).

Para configurar as interfaces de rede do servidor em nuvem, deve-se permitir que haja comunicação na porta definida arbitrariamente, neste caso, 1883, bem como o protocolo permitido de comunicação, conforme a Figura 5.

Figura 5 – Regras de rede de saída do servidor AWS.



Fonte: Do Autor (2025).

O acesso do servidor em nuvem é feito via comando SSH (*Secure Shell*), utilizando um arquivo de chave privada disponibilizado pela AWS. Tal acesso é permitido com a chave privada no computador em que vai acessar a instância EC2. No Código 3.5 é apresentado como é realizada essa comunicação.

Código 3.5 – Comando para acesso ao servidor AWS via SSH com chave privada.

```
$ ssh -i </caminho/para/chave.pem> ubuntu@<IP_publico_EC2>
```

3.2.1 Instalação do Broker MQTT e Banco de Dados

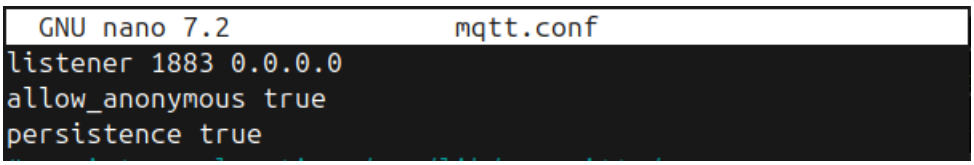
Após acessar a instância, o primeiro passo é configurar o *broker* MQTT central para receber dados dos *containers* de simulação e do nó de borda. Para isso, foi instalada a biblioteca *Mosquitto* e suas dependências. No Código 3.6 é apresentado como foi realizado esse processo.

Código 3.6 – Comandos para instalação do *Mosquitto* no servidor AWS.

```
$ sudo apt install mosquitto mosquitto-clients
```

Com o Mosquitto instalado, faz-se necessário definir a porta de comunicação e a disponibilidade de acesso por parte de outros dispositivos. Neste caso, utilizou-se a porta 1883 e foi configurado o acesso anônimo para simplificar a comunicação dos *containers* responsáveis pela geração dos sinais, conforme a Figura 6.

Figura 6 – Arquivo de configuração do MQTT no servidor AWS.



Fonte: Do Autor (2025).

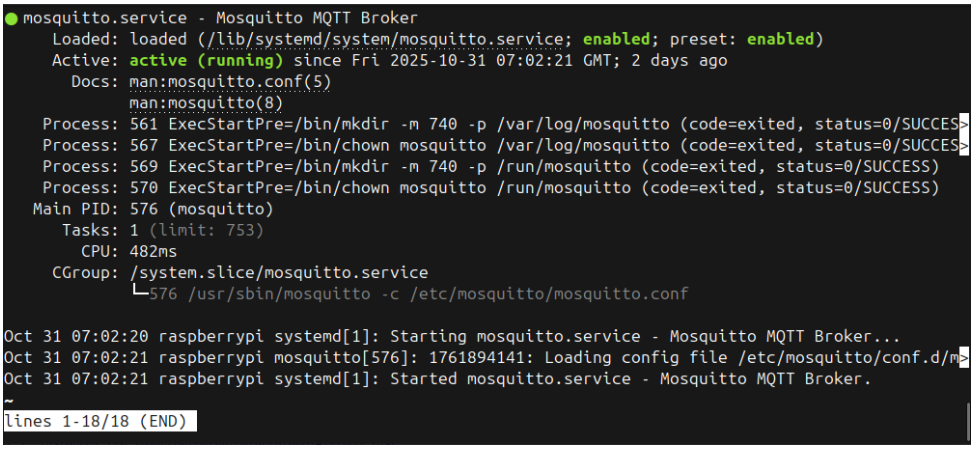
Por fim, o serviço foi habilitado para iniciar automaticamente com o sistema operacional, utilizando o gerenciador de serviços *systemctl*, conforme o Código 3.7.

Código 3.7 – Comandos para iniciar e habilitar o serviço *Mosquitto* no servidor AWS.

```
$ sudo systemctl enable mosquitto.service
$ sudo systemctl start mosquitto.service
```

O status de execução pode ser visualizado pelo comando *status* do mesmo sistema, conforme a Figura 7.

Figura 7 – Serviço *Mosquitto* inicializado em *background* no servidor AWS.



Fonte: Do Autor (2025).

3.2.1.1 Configuração de Visualização de Dados

Para permitir a visualização dos dados no *Grafana*, faz-se necessário instalar e inicializar o serviço e habilitá-lo da mesma forma que o *Mosquitto*, conforme o Código 3.8.

Código 3.8 – Comando para instalar e habilitar o Grafana-server.

```
$ sudo apt install grafana-server
$ sudo systemctl enable grafana-server
$ sudo systemctl status grafana-server
```

Para ser possível visualizar os dados no servidor Grafana, no entanto, faz-se necessário a utilização de um banco de dados compatível. Neste trabalho, devido a sua simplicidade de implementação e alta utilização no mercado, optou-se por utilizar o banco de dados InfluxDB. Para isso, basta instalá-lo e habilitar seu processo de forma semelhante aos demais processos, conforme o Código 3.9.

Código 3.9 – Comando para instalar e habilitar o InfluxDB.

```
$ sudo apt install influxd
$ sudo systemctl enable influxd
$ sudo systemctl start influxd
```

O banco de dados por sua vez utiliza a porta 8086 para sua comunicação via API. Dessa forma, é preciso garantir que essa porta, assim como a porta padrão do serviço Grafana (porta 3000), estejam liberadas no *Security Group* da instância AWS. Com a instalação e as regras de rede configuradas, o servidor em nuvem está pronto para receber os dados MQTT, armazená-los no InfluxDB e permitir sua visualização através da interface do Grafana.

3.3 CONFIGURAÇÃO DO RASPBERRYPI

Para a implementação do nó de borda, visando uma abordagem mais realista, optou-se por utilizar um dispositivo físico, a saber, um RaspberryPi Model 4, com 4 CPUs, 8 Gb de memória RAM, utilizando uma distribuição *Debian trixie* versão 13.1. Para isso, primeiro foi necessário configurar o ambiente de execução do dispositivo.

No contexto desta arquitetura, o Raspberry Pi atua como um nó de borda, responsável por coletar dados dos *containers*, processá-los com o fim de obter métricas dos sinais obtidos, como média, desvio padrão e predições do algoritmo LSTM e, em seguida, publicar os resultados no broker MQTT hospedado no servidor AWS.

Para que o Raspberry Pi possa se comunicar com o meio e com o servidor em nuvem, é essencial que os pacotes clientes do Mosquitto estejam instalados, permitindo que ele se conecte e interaja com o serviço MQTT central através da porta 1883. O processo para instalação dessa aplicação são semelhantes aos realizados no servidor AWS.

Com isso, basta iniciar o algoritmo responsável por captar os valores transmitidos a ele via MQTT, processá-los, e enviar via MQTT para o servidor em nuvem, atuando como um nó de borda. O processamento utilizado no RaspberryPi, bem como o funcionamento da transmissão dos dados para a nuvem, são explicados na seção 3.4.

3.4 IMPLEMENTAÇÃO DO ALGORITMO LSTM

A predição dos sinais provenientes dos sensores foi realizada por meio de um modelo de rede neural recorrente do tipo LSTM (*Long short-term memory*), implementado em Python utilizando a biblioteca `TensorFlow/Keras`, e aplicado no nó de borda. Esse tipo de arquitetura é amplamente empregado na modelagem de séries temporais devido à sua capacidade de capturar dependências de curto e longo prazo, mesmo em sinais com comportamento dinâmico e ruído inerente.

3.4.1 Estrutura e funcionamento do modelo

O sistema foi projetado para operar de forma totalmente *online*, processando continuamente novas amostras recebidas via MQTT. Cada sensor introduz ao modelo LSTM os valores de média, desvio padrão, máximo e mínimo. As medições mais recentes são armazenadas em uma janela deslizante de tamanho fixo contendo as últimas N amostras. Essa janela é utilizada para calcular a média, o desvio padrão, máximo e mínimo das N amostras.

Para o treinamento supervisionado, os dados são organizados em pequenas sequências de entrada e saída. Cada sequência de *input* é composta por um conjunto de L amostras consecutivas, e o objetivo da rede é aprender uma função não linear capaz de estimar o próximo valor da série temporal com base em seu histórico recente. Assim, o modelo gerado visa minimizar o erro entre o valor real e o valor previsto.

3.4.2 Configuração da arquitetura e treinamento

A arquitetura desenvolvida é composta por uma única camada LSTM contendo 32 unidades e uma camada densa (*Dense*) com um único neurônio de saída, responsável por gerar as predições da média, desvio padrão, máximo e mínimo dos sensores. Essa configuração foi escolhida por equilibrar simplicidade e desempenho, uma vez que deve ser implementada não só no servidor em nuvem, mas também no nó de borda, sendo suficiente para capturar a dinâmica temporal dos sinais analisados. O Código 3.10 define, então, a estrutura do modelo.

Código 3.10 – Definição do modelo LSTM.

```
model = Sequential([
LSTM(32, activation='linear', input_shape=(look_back, 1)),
Dense(1)
])
```

Durante a execução, a cada 50 novas amostras recebidas, o modelo é atualizado de forma incremental. Esse processo de treinamento permite que a rede se adapte gradualmente a mudanças no comportamento dos sensores, sem a necessidade de reiniciar o treinamento, o que seria inviável em sistemas com fluxo contínuo de dados.

Após o treinamento, o modelo é utilizado para gerar previsões futuras do meio. Como entrada, recebe-se valores de média, desvio padrão, máximo e mínimo de cada sensor dentro da janela. A partir destes valores, calcula-se estimações das mesmas métricas das próximas amostras. Neste trabalho, optou-se por prever as métricas dos dois próximos conjuntos amostrais de 50 amostras.

3.5 DEFINIÇÃO DAS MÉTRICAS AVALIADAS

Para avaliar a qualidade de serviço da rede em todos os cenários, são utilizadas algumas métricas e indicadores de rede, a saber: latência, *Jitter* e *Throughput*.

A latência é um dos principais indicadores da qualidade de uma transmissão de dados, uma vez que se refere a diferença temporal entre o tempo de envio de uma mensagem e o tempo de chegada da mesma mensagem a um determinado receptor, ou seja, o atraso no envio de um dado de um ponto a outro. Assim, para mensurar a latência nos testes realizados, será coletado o *timestamp*, isto é, o registro temporal antes da mensagem ser transmitida, e o *timestamp* de quando a mensagem for recebida pelo receptor.

Com essas informações, a latência para o i-ésimo pacote é dada pela Equação (1), onde t_{tx} e t_{rx} representam o tempo em que a mensagem foi transmitida e recebida, respectivamente.

$$L_i = t_{tx} - t_{tx} \tag{1}$$

O *Jitter*, por sua vez, é o desvio padrão da latência, conforme a Equação (2), com $\bar{L}$ representando o valor médio da latência para o conjunto de N amostras.

$$J = \sqrt{\frac{\sum_{i=1}^{N}(L_i - \bar{L})^2}{N-1}} \tag{2}$$

Por fim, o *throughput*, ou vazão, é mensurado através da razão entre o volume de dados recebidos pelo receptor em um determinado período de tempo, conforme a Equação (3)

$$T = \frac{V_{\text{dados}}}{t_{\text{total}}} \tag{3}$$

onde V_{dados} representa o volume de dados (em bits) transferidos com sucesso e t_{total} denota o tempo total (em segundos) necessário para a transferência de V_{dados}. Neste trabalho é considerado o volume total de cada pacote para compor V_{dados}.

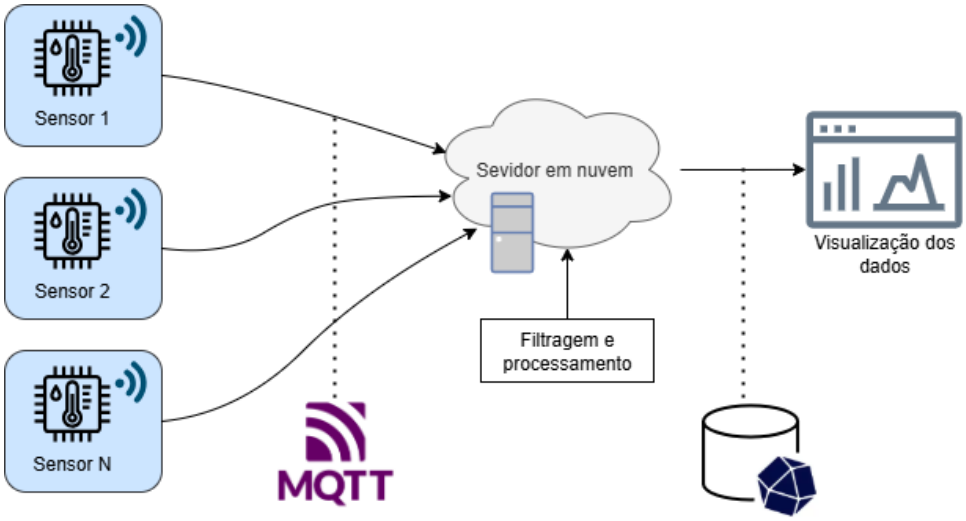
3.6 IMPLEMENTAÇÃO DOS CENÁRIOS DE TESTE

Para a implementação proposta, primeiro faz-se necessário a definição dos diferentes cenários que serão avaliados. Assim, definiu-se três topologias de modo a abranger as principais situações reais, que podem ser visualizadas nas subseções a seguir.

3.6.1 Cenário 1: Comunicação direta com o servidor em nuvem

Este cenário contempla uma rede MMTC sem o nó de borda, isto é, todos os dados gerados pelo meio são enviados para o servidor em nuvem para ser processados e utilizados, conforme pode-se obsevrar na Figura 8. Neste ambiente, tem-se como objetivo estabelecer uma referência para comparação.

Figura 8 – Diagrama do cenário 1.

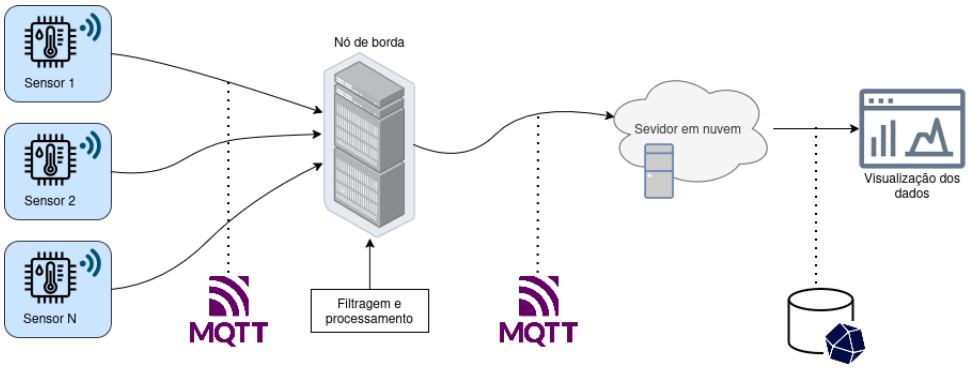


Fonte: Do Autor (2025).

3.6.2 Cenário 2: Comunicação dos sensores com o nó de borda e servidor em nuvem

Este cenário contempla uma rede MMTC com o nó de borda, isto é, todos os dados gerados pelo meio são enviados primeiramente para o computador de borda, onde serão calculadas todas as *features*, enviando apenas os valores finais para o servidor em nuvem, conforme pode-se obsevrar na Figura 9.

Figura 9 – Diagrama do cenário 2.



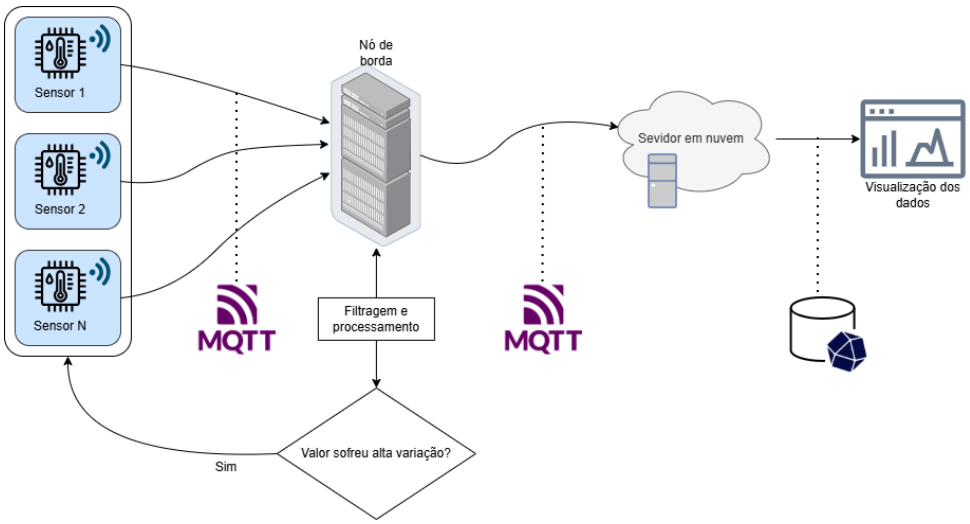
Fonte: Do Autor (2025).

3.6.3 Cenário 3: Comunicação com *feedback* para o ambiente

Por fim, neste cenário será avaliada uma aplicação de atuação por parte do nó de borda para o meio, caso seja identificado alguma alteração indesejada das *features*. Este

cenário tem como principal objetivo avaliar a velocidade do tempo de resposta no *feedback* para o meio. O diagrama do cenário 3 pode ser visualizado na Figura 10.

Figura 10 – Diagrama do cenário 3.



Fonte: Do Autor (2025).

4 RESULTADOS

Com o ambiente de simulação preparado tanto no RaspberryPi quanto no AWS, pode-se então iniciar as simulações. Esta seção tem como objetivo analisar os resultados obtidos em cada um dos cenários previamente definidos.

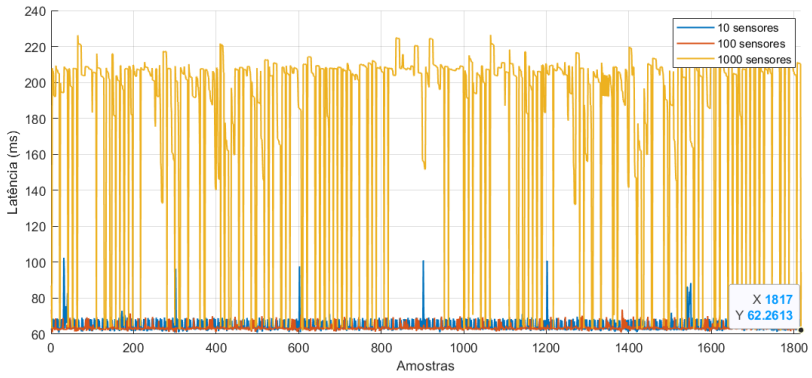
4.1 CENÁRIO 1

4.1.1 Cenário 1: Sem a utilização do TC

Primeiramente, foram realizadas medições sem qualquer variação no ambiente, com conjuntos de 10, 100 e 1000 sensores, conectados diretamente ao servidor em nuvem, enviando valores em intervalos de 2 segundos de forma paralela. Devido às variações no tempo de execução em cada *container*, a rede é populada com informação de forma semelhante a sensores reais.

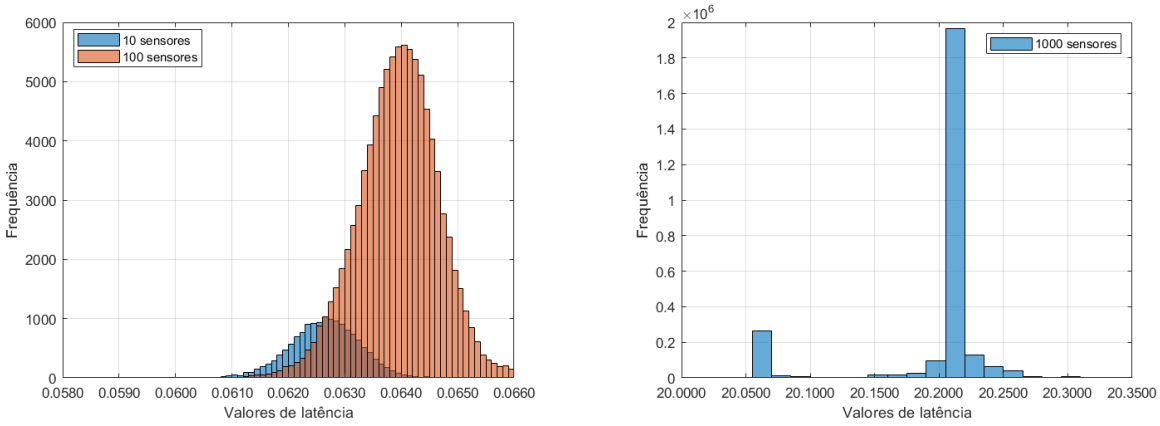
Os resultados de latência e *throughput* em função do tempo podem ser visualizados nas Figuras 11 e 13, enquanto a distribuição dos valores de latência pode ser visualizado nas Figuras 12 (a) e 12 (b). Além disso, os valores médios gerais de latência, *jitter* e *throughput* podem ser visualizados na Tabela 1. Pode-se observar que a latência é diretamente afetada com o aumento do número de dispositivos transmissores, com valores médios elevados e um aumento notável no desvio padrão, comportamento confirmado ao observar os histogramas de latência. Ao observar o gráfico do *throughput*, nota-se de forma muito evidente, o efeito direto do aumento de dispositivos transmissores no consumo de banda da rede, chegando a um valor médio de $1,129$ Mpbs com a utilização de 1000 sensores.

Figura 11 – Cenário 1: curvas de latência sem a utilização do TC.



Fonte: Do Autor (2025).

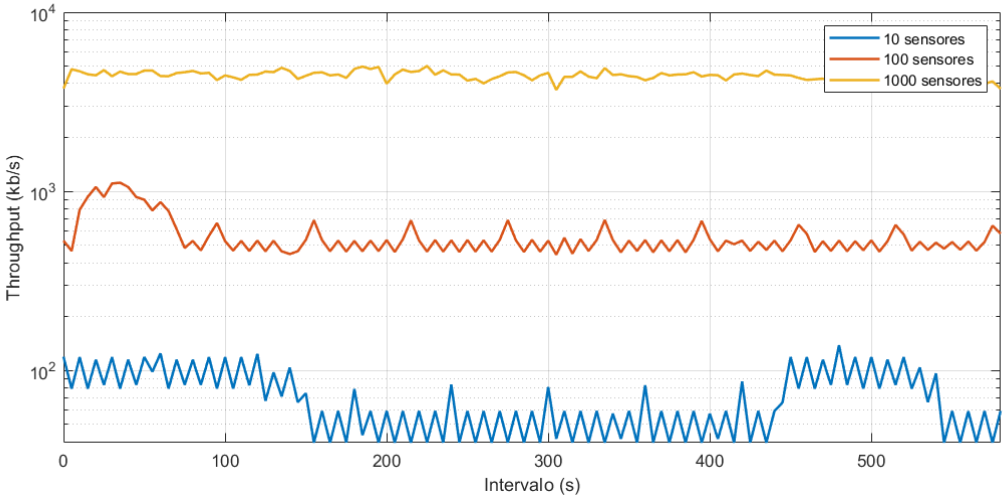
Figura 12 – Cenário 1: histogramas de latência sem a utilização do TC para diferentes quantidades de sensores.



(a) 10 e 100 sensores. (b) 1000 sensores.

Fonte: Do Autor (2025).

Figura 13 – Cenário 1: curvas de *throughput* sem a utilização do TC.



Fonte: Do Autor (2025).

Tabela 1 – Cenário 1: métricas obtidas sem a utilização do TC.

N° sensores	**Latência média**	***Jitter***	***Throughput* médio**
10	62,3 ms	7,76 ms	40,3915 kbps
100	64,3 ms	7,49 ms	201,3072 kbps
1000	202,1 ms	9,45 ms	1,129 Mbps

4.1.2 Cenário 1: Utilização do TC para cenários adversos

Com o fim de observar o comportamento de uma rede sob efeitos adversos, utilizou-se o comando TC. Neste cenário, considerou-se uma lentidão intrínseca na comunicação

com o servidor em nuvem, ocasionando um alto atraso na comunicação. Além disso, considerou-se uma comunicação instável e oscilante, representada por um alto valor de *jitter* e perda de pacotes de dados. Por fim, para considerar uma limitação de capacidade da rede, limitou-se também o limite de banda consumida. Os valores utilizados, escolhidos de forma arbitrária, podem ser visualizados na Tabela 2. Esses valores foram definidos de forma a se adequar a situações reais de intermitência e comprometimento da rede. Assim, o comando TC utilizado no dispositivo de saída, nesse caso, na transmissão dos *containers*, pode ser visualizado no Código 4.1.

Tabela 2 – Cenário 1: parâmetros de simulação de rede configurados com o comando TC.

Métrica	**Variação**
Latência média	100 ms
Jitter	20 ms
Limite de *Throughput*	500 kbps
Perda de pacotes	2%

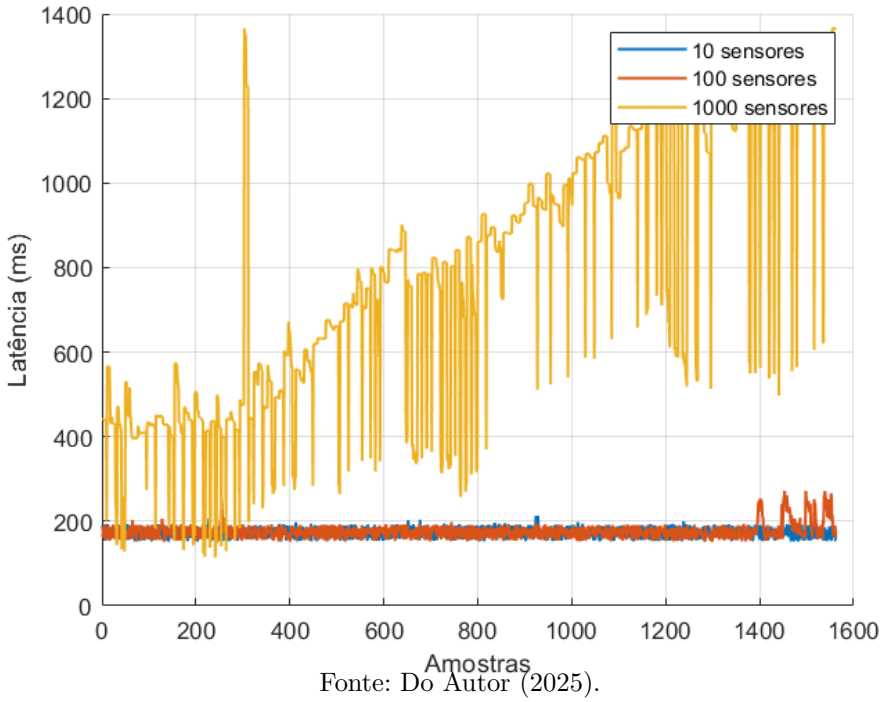
Código 4.1 – Comando para simular o ambiente utilizando a ferramenta TC.

```
1 $ sudo tc qdisc add dev <interface_de_rede> parent 1:1 handle 2: netem
      delay 100ms 20ms loss 2%
2 $ sudo tc qdisc add dev <interface_de_rede> root handle 1: htb default
      1
3 $ sudo tc class add dev <interface_de_rede> parent 1: classid 1:1 htb
     rate 500kbit
```

Os resultados obtidos com a utilização do TC, no que tange a latência em função do tempo, bem como seu histograma e o *throughput* em função do tempo, podem ser visualizados nas Figuras 14, 15 e 16, respectivamente. Por fim, os valores médios gerais de latência, *jitter* e *throughput* pode ser visualizada na Tabela 3.

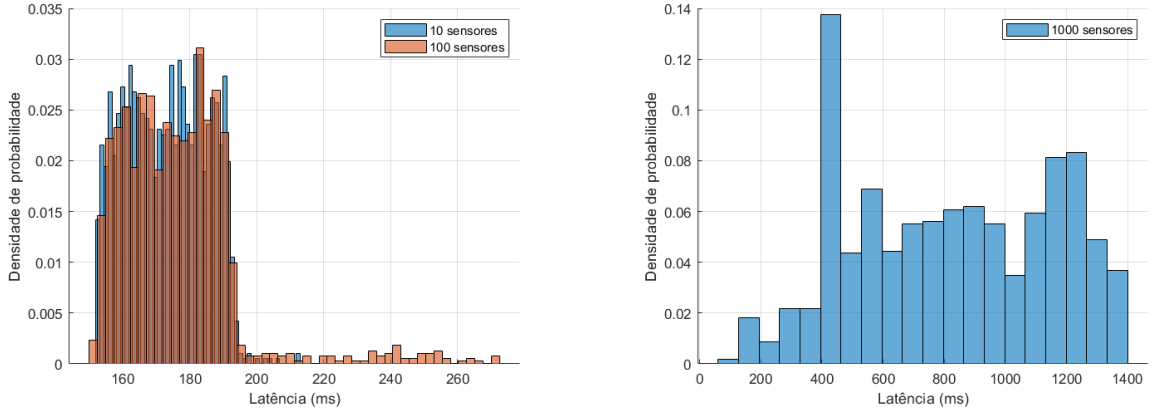
Observa-se que a QoS é fortemente comprometida quando o sistema opera com 1000 sensores. Nessa escala, o congestionamento na rede torna-se tão elevado que as mensagens passam a levar vários segundos para serem entregues. A análise do *throughput* mostra que toda a banda disponível, limitada neste caso a 500 kbps, é rapidamente esgotada, levando a rede à saturação. Como consequência, tem-se o enfileiramento de pacotes, aumento da latência, e maior taxa de perda de pacotes. Quando esses fatores são somados ao tempo de processamento necessário para disponibilizar as *features* ao usuário final, torna-se evidente a relevância da computação em borda como estratégia para reduzir a latência e aumentar a eficiência global do sistema.

Figura 14 – Cenário 1: curvas de latência com a utilização do TC.



Fonte: Do Autor (2025).

Figura 15 – Cenário 1: histogramas de latência com a utilização do TC para diferentes quantidades de sensores.

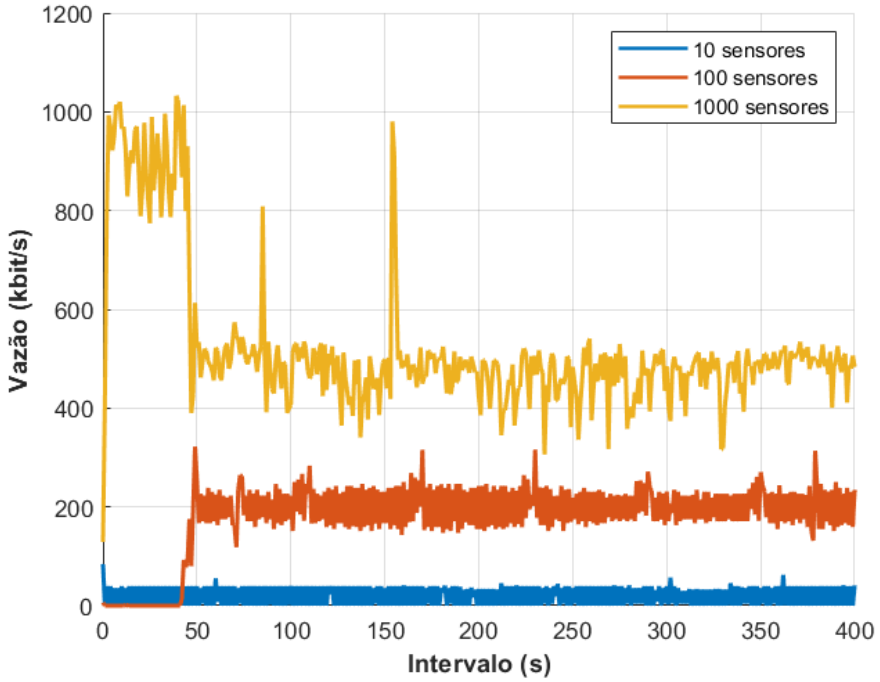


(a) 10 e 100 sensores.

(b) 1000 sensores.

Fonte: Do Autor (2025).

Figura 16 – Cenário 1: curvas de *throughput* com a utilização do TC.



Fonte: Do Autor (2025).

Tabela 3 – Cenário 1: métricas obtidas com a utilização do TC.

N° sensores	**Latência média**	***Jitter***	***Throughput* médio**
10	173, 07 ms	11, 80 ms	49, 44 kbps
100	180, 44 ms	11, 85 ms	202, 70 kbps
1000	818, 19 ms	329, 58 ms	525, 015 kbps

4.2 CENÁRIO 2

Tendo avaliado o comportamento da rede, realiza-se então as simulações utilizando um nó de borda entre os sensores e o servidor em nuvem. Neste cenário, a ação do nó de borda irá consistir na recepção dos dados dos sensores, processamento da predição das *features* futuras através da rede LSTM, e envio para o servidor em nuvem. Dessa forma, o papel do AWS será apenas o de receber as *features* e disponibilizar para o usuário.

4.2.1 Cenário 2: Sem a utilização do TC

De forma análoga ao primeiro cenário, primeiramente avalia-se o comportamento da rede MMTC sem realizar alterações de ambiente pelo comando TC, mas agora com a contribuição de um nó de borda. A captura dos pacotes pelo *Wireshark* e a coleta dos dados foram realizadas entre os sensores e o nó de borda, e também entre o nó de borda e o servidor em nuvem. Os resultados de latência em função do tempo podem ser visualizados

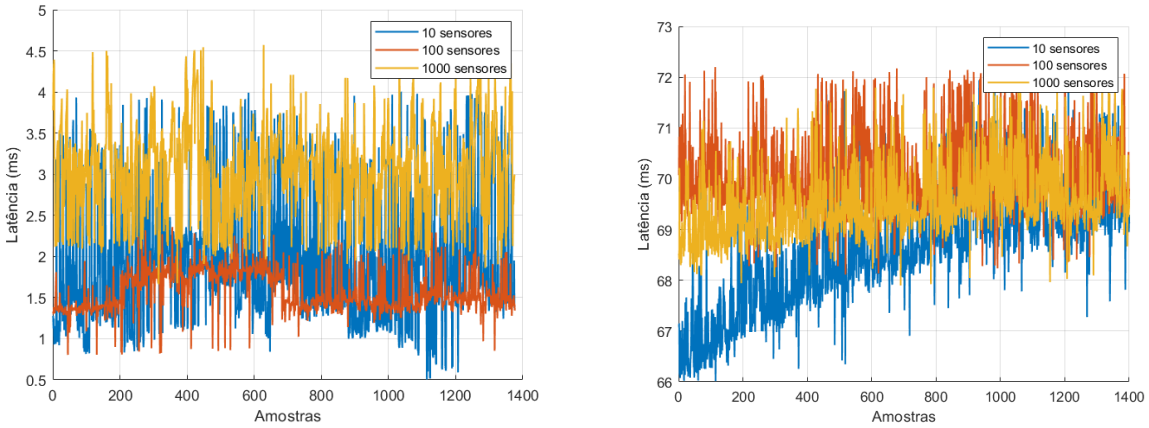
na Figura 17, bem como a distribuição dos valores de latência pode ser visualizado na Figura 18, enquanto o *throughput* pode ser visualizado na Figura 19. Por fim, os valores médios dessas métricas podem ser visualizados na Tabela 4.

Ao observar o comportamento da latência, pode-se notar que há uma drástica redução no atraso das mensagens entre o meio e o nó de borda. Isso ocorre, pois o nó de borda se encontra fisicamente próximo aos sensores, tornando a comunicação muito mais rápida. Além disso, pode-se também observar que há uma estabilização no valor de latência do nó de borda para o servidor em nuvem, mesmo com o aumento da quantidade de sensores, ficando em torno de 70 ms.

Vale ressaltar que a comunicação entre os sensores e o nó de borda é passível de variações não controladas, causadas por limitações de *hardware*, uma vez que o nó de borda foi implementado em um dispositivo físico. Tal variabilidade pode ser visualizada nos valores médios de *jitter*, por exemplo, onde a simulação utilizando 100 sensores resultou em um desvio padrão inferior a simulação com 10 sensores.

Ademais, ao analisar a relação do *throughput* em função do tempo e da quantidade de sensores, observa-se que o comportamento da rede entre o meio e o nó de borda é bem semelhante ao Cenário 1, onde a quantidade de sensores afeta diretamente o consumo de banda, mas que se mantém estável ao analisar a saída dos dados para o servidor em nuvem. Isso porque mesmo com o aumento maior do número de sensores, os dados são enviados apenas a cada 50 amostras, ao invés de cada amostra ser enviada de forma ininterrupta.

Figura 17 – Cenário 2: curvas de latência sem a utilização do TC.

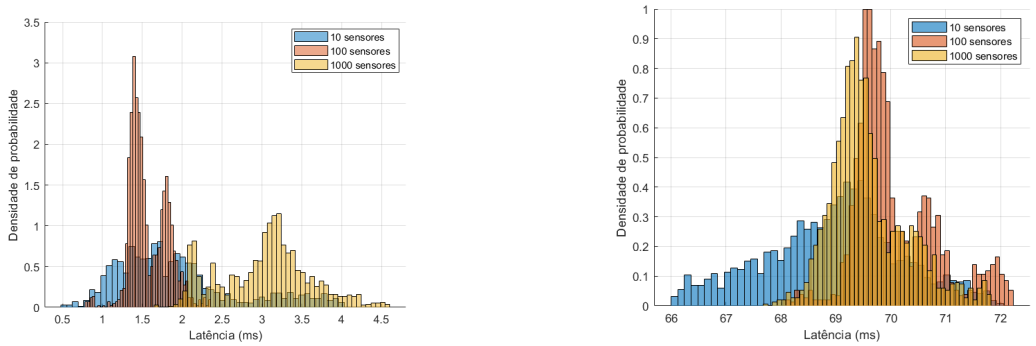


(a) Entre os sensores e o nó de borda.

(b) Entre o nó de borda e o AWS.

Fonte: Do Autor (2025).

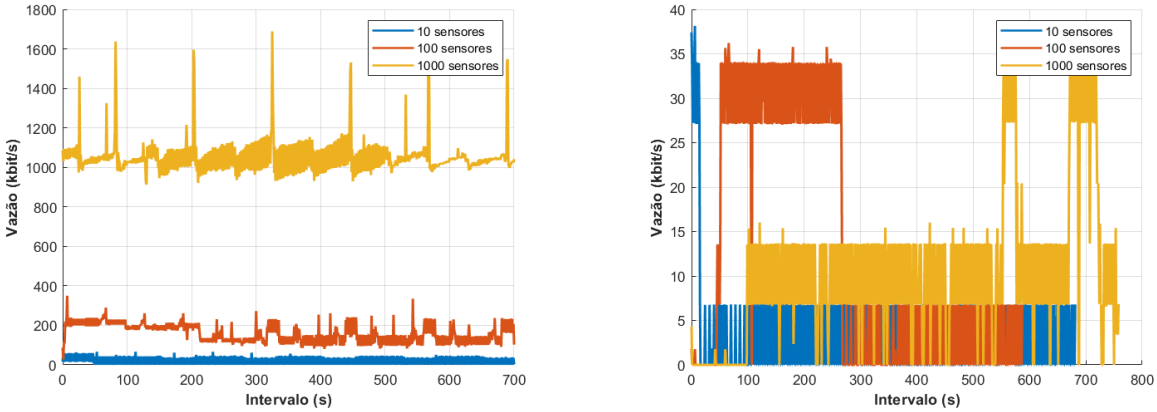
Figura 18 – Cenário 2: histogramas de latência sem a utilização do TC para diferentes quantidades de sensores.



(a) Entre os sensores e o nó de borda. (b) Entre o nó de borda e o AWS.

Fonte: Do Autor (2025).

Figura 19 – Cenário 2: curvas de *throughput* do meio para o nó de borda, e do nó de borda para a nuvem, sem a utilização do TC.



(a) Entre os sensores e o nó de borda. (b) Entre o nó de borda e o AWS.

Fonte: Do Autor (2025).

Tabela 4 – Cenário 2: métricas obtidas sem a utilização do TC.

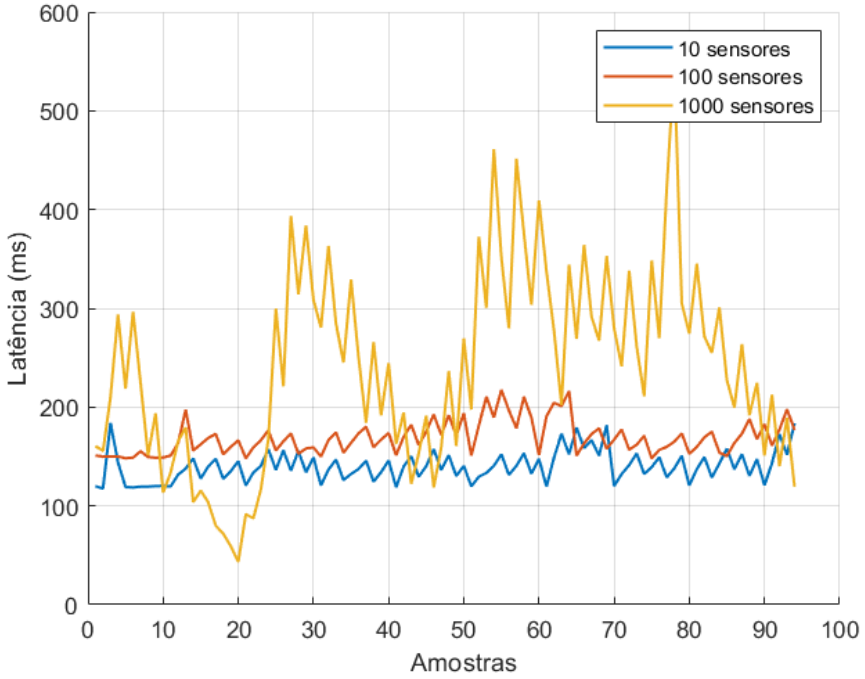
N° sensores	**Latência média**	***Jitter***	***Throughput* médio**
Entre os sensores e o nó de borda			
10	1, 92 ms	0, 76 ms	22, 53 kbps
100	1, 53 ms	0, 26 ms	154, 34 kbps
1000	3, 08 ms	0, 58 ms	1, 05 Mbps
Entre o nó de borda e o AWS			
10	69, 00 ms	3, 20 ms	2, 70 kbps
100	70, 06 ms	3, 74 ms	22, 17 kbps
1000	73, 79 ms	3, 71 ms	20, 10 kbps

4.2.2 Cenário 2: Utilização do TC para cenários adversos

Para validar a influência do nó de borda em cenários adversos, realizou-se a mesma simulação de ambiente abordada no Cenário 1, cujos parâmetros podem ser visualizados na Tabela 2. O resultado da latência entre o nó de borda e o AWS pode ser visualizado na Figura 20, os histogramas de latência em função da quantidade de sensores podem ser verificados na Figura 21 e o *throughput* pode ser observado na Figura 22. Além disso, os valores médios da rede podem ser visualizados na Tabela 5.

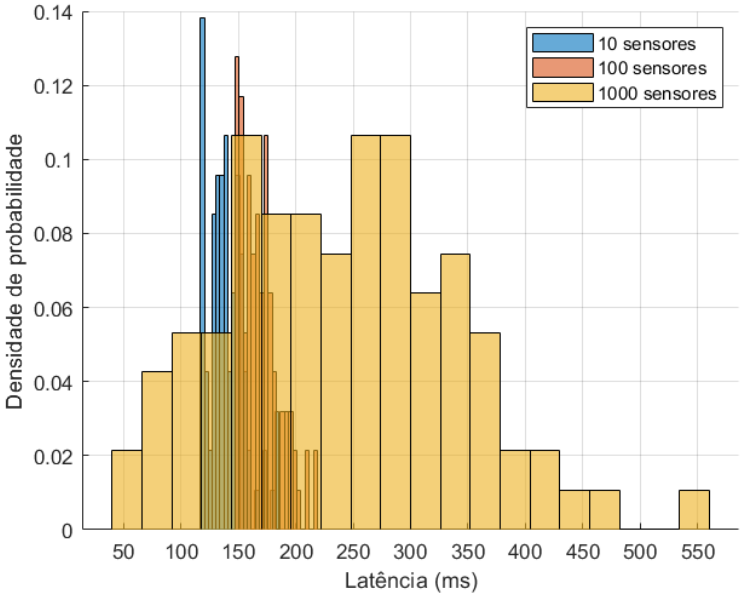
Ao analisar o comportamento da latência nesse cenário, nota-se um aumento significativo tanto no valor médio quanto na variabilidade, especialmente nos casos com maior número de sensores, chegando a alcançar uma latência média de $238,09$ e um *jitter* de $107,64$ ms no cenário com 1000 sensores. Ainda assim, apesar do acréscimo expressivo no atraso e no *jitter*, o sistema permaneceu estável, sem acúmulo de mensagens. Quanto ao *throughput*, observa-se que sua variação ao longo do tempo acompanha a quantidade de sensores, mas sem indicar saturação. Na simulação com 100 sensores, a vazão estabiliza em torno de 268 kbps, enquanto no cenário com 1000 sensores o *throughput* atinge aproximadamente 370 kbps. Esses valores mostram que, mesmo com o aumento da carga, a capacidade de transmissão da rede não foi completamente esgotada.

Figura 20 – Cenário 2: curvas de latência entre o nó de borda e o AWS com a utilização do TC.



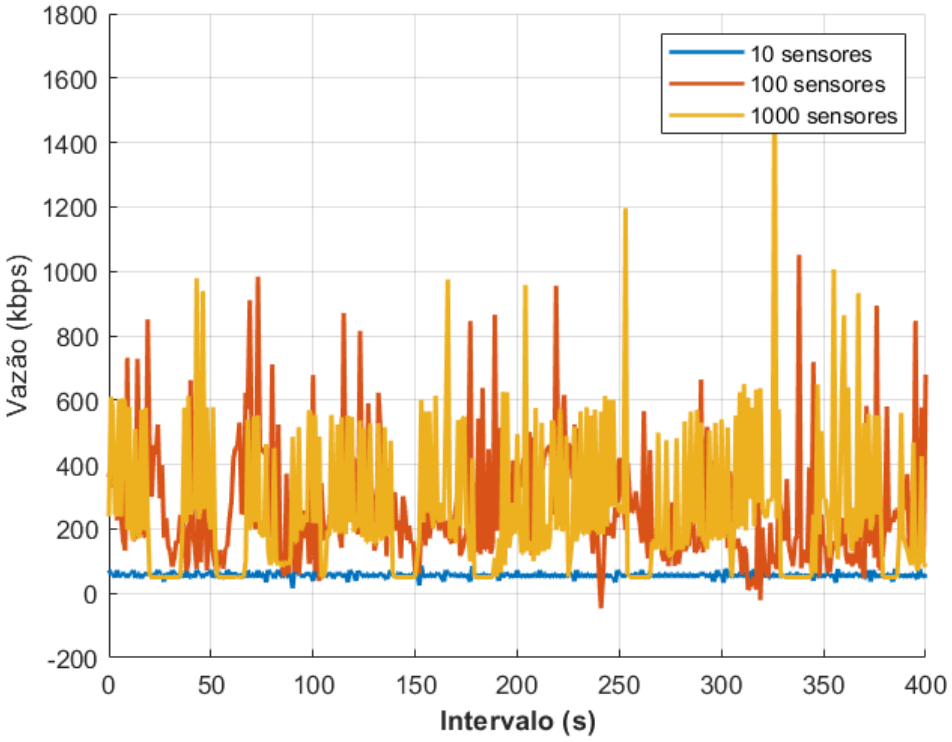
Fonte: Do Autor (2025).

Figura 21 – Cenário 2: histogramas de latência com a utilização do TC para diferentes quantidades de sensores.



Fonte: Do Autor (2025).

Figura 22 – Cenário 2: curvas de *throughput* com a utilização do TC entre o nó de borda e o AWS.



Fonte: Do Autor (2025).

Tabela 5 – Cenário 2: métricas obtidas com a utilização do TC entre o nó de borda e a nuvem.

N° sensores	**Latência média**	***Jitter***	***Throughput* médio**
10	140,93 ms	16,02 ms	76,02 kbps
100	169,03 ms	17,11 ms	268,94 kbps
1000	238,09 ms	107,64 ms	370,30 kbps

4.3 CENÁRIO 3

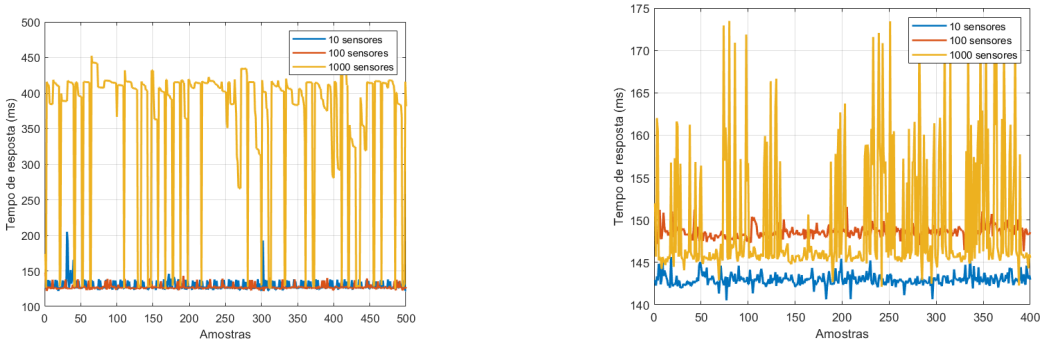
Por fim, após avaliar a influência do nó de borda na QoS da rede, analisa-se outra aplicação prática dos computadores de borda: a redução do tempo de resposta ao ambiente. Nesse cenário, ilustrado na Figura 10, busca-se verificar a capacidade do nó de borda de atuar rapidamente em situações de anomalia.

O objetivo é comparar a velocidade de resposta do sistema com nó de borda em relação a um ambiente conectado exclusivamente à nuvem. Considera-se, para o servidor em nuvem, que o tempo de resposta corresponde ao *round-trip time*, isto é, o intervalo necessário para que a mensagem vá e retorne ao ponto de origem. Dessa forma, serão registrados os tempos associados a cada etapa de envio e recepção de dados, permitindo determinar o tempo total de resposta.

Primeiramente avalia-se o comportamento da rede MMTC com a contribuição de um nó de borda, sem realizar alterações de ambiente pelo comando TC. Nas Figuras 23 (a) e 23 (b), pode-se observar o tempo de resposta entre os sensores e o servidor em nuvem, e o tempo de resposta entre os sensores e o nó de borda, respectivamente. Os histogramas do tempo de resposta podem ser visualizados nas Figuras 24 e 25, e os valores médios gerais podem ser visualizados na Tabela 6.

Ao analisar o tempo de resposta de forma comparativa, torna-se evidente a melhoria na rapidez com que informações podem retornar ao meio, com o auxílio de um nó de borda em cenários mais críticos. Enquanto que o tempo de resposta quando conectado ao AWS resultou em uma média de 404,63 ms no cenário com 1000 sensores, a atuação do nó de borda foi capaz de reduzir para 149,72 ms de média. Em contrapartida, em cenários com menor quantidade de sensores, a utilização do nó de borda pode acarretar em um incremento de tempo de transporte e processamento, uma vez que estabelece um dispositivo intermediário na comunicação.

Figura 23 – Cenário 3: histogramas de tempo de resposta (*round-trip*).

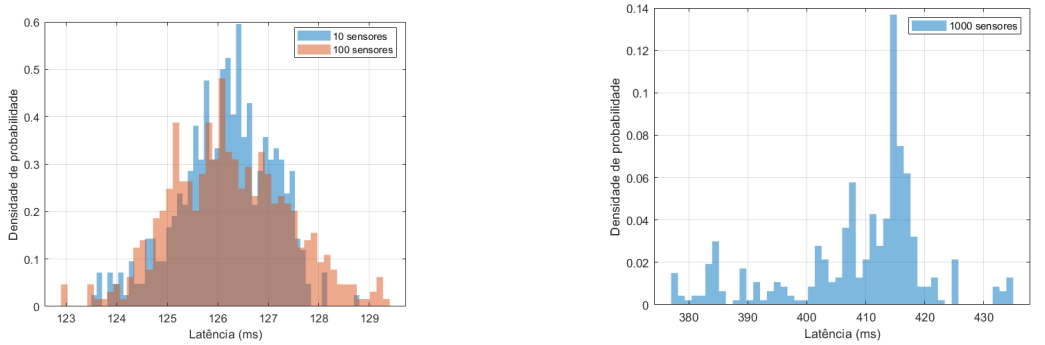


(a) Tempo de resposta entre os sensores e o AWS.

(b) Tempo de resposta entre os sensores e o nó de borda.

Fonte: Do Autor (2025).

Figura 24 – Cenário 3: histogramas de tempo de resposta (*round-trip*) entre o sensor e o AWS.

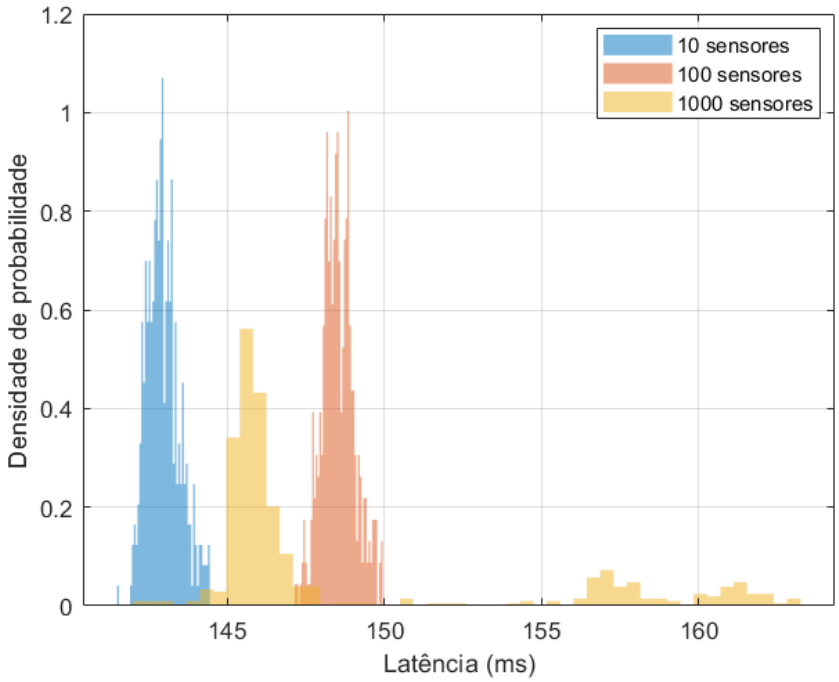


(a) 10 e 100 sensores.

(b) 1000 sensores.

Fonte: Do Autor (2025).

Figura 25 – Cenário 3: histograma de tempo de resposta (*round-trip*) entre o sensor e o nó de borda.



Fonte: Do Autor (2025).

Tabela 6 – Cenário 3: métricas do tempo de resposta.

N° sensores	**Tempo de resposta médio**	**Desvio padrão**
Entre os sensores e o AWS		
10	126, 02ms	1, 08ms
100	135, 09ms	3, 34ms
1000	404, 63ms	21, 79ms
Entre os sensores e o nó de borda		
10	143, 01ms	0, 7ms
100	148, 61ms	0, 7ms
1000	149, 72ms	7, 21ms

4.4 ANÁLISE DOS RESULTADOS

Ao analisar comparativamente os três cenários propostos, torna-se evidente a influência da computação de borda na melhoria da qualidade de serviço em cenários MMTC. No primeiro cenário, em que os sensores se comunicam diretamente com o servidor em nuvem, observou-se que a latência e o *jitter* aumentam significativamente à medida que cresce a quantidade de dispositivos conectados. Em condições normais, a latência média passou de aproximadamente 62 ms com 10 sensores para mais de 200 ms com 1000 sensores, acompanhada por um aumento expressivo no *throughput*, que atingiu cerca de 1, 129 Mbps. Esse comportamento indica pressão crescente sobre a infraestrutura de rede, mesmo

sem a introdução de condições adversas. Quando aplicadas restrições pelo comando TC, simulando atraso adicional, *jitter* elevado, perda de pacotes e limitação de banda a 500 kbps, o impacto foi ainda mais severo: a latência média ultrapassou 800 ms e o *jitter* chegou a 329 ms no cenário com 1000 sensores, evidenciando saturação e enfileiramento de pacotes.

No segundo cenário, com a inserção do nó de borda, os ganhos foram substanciais, sobretudo em ambientes de alta densidade e sob condições adversas. Em situações normais, a comunicação entre sensores e borda apresentou latências muito mais baixas, variando entre $1,5$ ms e 3 ms, enquanto o enlace entre borda e nuvem manteve-se estável em torno de 70 ms, mesmo com 1000 sensores. Além disso, o tráfego enviado à nuvem foi drasticamente reduzido, pois apenas as *features* processadas localmente eram transmitidas, limitando o *throughput* a cerca de 20 kbps, contra mais de 1 Mbps no cenário sem borda. Essa redução de carga no núcleo da rede contribui para maior previsibilidade e estabilidade da rede. Em condições adversas, embora tenha ocorrido aumento na latência e no *jitter*, os valores permaneceram significativamente inferiores aos observados no primeiro cenário. Com 1000 sensores, a latência média entre borda e nuvem foi de aproximadamente 238 ms, e o *throughput* estabilizou em torno de 370 kbps, abaixo do limite imposto pelo TC, evitando saturação completa e colapso comunicacional.

O terceiro cenário reforça a relevância da computação de borda em aplicações que exigem resposta rápida ao ambiente. Ao comparar o tempo de resposta entre sensores e nuvem com o tempo entre sensores e borda, verificou-se que, em alta densidade, a borda reduziu o tempo médio de 404 ms para cerca de 149 ms, representando uma diminuição superior a 63%. Essa característica é essencial para sistemas críticos, como controle industrial e automação, nos quais atrasos podem comprometer a operação.

De forma geral, os resultados demonstram que a computação de borda atua como elemento fundamental para garantir escalabilidade, resiliência e eficiência em redes com cenários MMTC. Ao reduzir o volume de dados transmitidos à nuvem, estabilizar métricas de latência e *jitter* sob condições adversas e encurtar tempos de resposta em aplicações críticas, a borda se apresenta como solução estratégica para evitar o colapso comunicacional em cenários massivos. Por outro lado, em cargas leves, seus benefícios são mais relacionados à arquitetura e à economia de recursos do que à melhoria perceptível da QoS.

5 CONCLUSÃO

O presente trabalho teve como objetivo investigar o impacto do uso de computação em borda na comunicação entre dispositivos inteligentes em cenários MMTC, avaliando seu desempenho quanto à latência, *jitter* e vazão (*throughput*). Para isso, foi desenvolvido um ambiente completo de simulação e coleta de métricas, envolvendo sensores virtuais, um nó de borda implementado em *Raspberry Pi* e um servidor em nuvem, integrados por meio de uma infraestrutura de comunicação MQTT.

Os experimentos realizados evidenciaram que o aumento do número de sensores intensifica o congestionamento da rede, afetando diretamente a QoS. Em cenários de alta densidade de dispositivos, observou-se um crescimento significativo nos valores de latência e *jitter*, enquanto as simulações de condições adversas demonstraram o colapso da comunicação direta entre sensores e servidor em nuvem, resultando na completa degradação da QoS no cenário com mil dispositivos conectados. Ademais, a análise da vazão mostrou que o fluxo de dados típico de cenários MMTC pode saturar totalmente a banda disponível, ocasionando na degradação da qualidade da transmissão dos dados.

A integração da computação em borda demonstrou-se fundamental para mitigar esses efeitos. Os resultados indicam que o processamento local reduziu o tempo de resposta, diminuiu a dependência da nuvem e reduziu o tempo de resposta (*round-trip time*) ao ambiente. Até mesmo em cenários adversos, a computação de borda foi capaz de impedir o colapso da rede, mantendo a vazão abaixo da taxa de saturação. Em síntese, os resultados obtidos confirmam que a computação em borda é uma solução eficaz para reduzir latência, melhorar a escalabilidade e elevar a robustez das redes em cenários MMTC.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Sugere-se, como continuidade deste estudo, a realização de experimentos que incluam sensores físicos reais, em substituição parcial ou total aos sensores simulados em *containers Docker*. Essa abordagem permitiria avaliar o comportamento do sistema em condições industriais autênticas, considerando variáveis inerentes ao ambiente físico, como ruído, variações de *hardware*, etc. Dessa forma, seria possível validar e comparar o desempenho da arquitetura aqui proposta com dados provenientes de um ambiente operacional real.

Recomenda-se, adicionalmente, a condução de uma análise detalhada do consumo energético do nó de borda e no servidor em nuvem. Tal investigação poderia abranger cenários de operação contínua, modos de economia de energia, diferentes cargas de tráfego e variações no uso do algoritmo de predição. Com isso, seria possível quantificar o impacto energético da computação de borda em aplicações MMTC, contribuindo para a definição de políticas mais eficientes de alocação de recursos.

Outra sugestão consiste na realização de uma avaliação aprofundada dos mecanis-

mos de segurança e privacidade aplicáveis ao protocolo MQTT no contexto de comunicação entre o nó de broda e a nuvem. Propõe-se investigar o uso de criptografia TLS, autenticação por certificados digitais e políticas de controle de acesso, analisando o impacto desses mecanismos sobre as métricas de QoS observadas. Ademais, recomenda-se estudar estratégias para mitigar os efeitos de sobrecarga introduzidos por tais mecanismos, visando disponibilizar comunicações seguras sem comprometer requisitos de latência, *jitter* ou vazão.

Por fim, recomenda-se expandir a presente arquitetura por meio da implementação de técnicas de escalonamento automático de recursos na borda e na nuvem, utilizando ferramentas de orquestração dinâmica, como *Kubernetes* ou plataformas equivalentes. A inclusão de múltiplos nós de borda, instâncias distribuídas e políticas de migração de cargas permitiria analisar o comportamento do sistema em cenários de alta demanda e elevada densidade de dispositivos, demonstrando a escalabilidade da solução e possibilitando estudos comparativos entre diferentes estratégias de orquestração e alocação de recursos.

REFERÊNCIAS

AI, Yuan; PENG, Mugen; ZHANG, Kecheng. Edge computing technologies for Internet of Things: a primer. **Digital Communications and Networks**, v. 4, n. 2, p. 77–86, 2018. ISSN 2352-8648.

AMAZON WEB SERVICES, INC. **Overview of Amazon Web Services**. [*S.l.*: *s.n.*], 2024. https://d0.awsstatic.com/whitepapers/aws-overview.pdf. Whitepaper oficial da Amazon Web Services. Disponível em: https://d0.awsstatic.com/whitepapers/aws-overview.pdf.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The Internet of Things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, 2010. ISSN 1389-1286.

BENNIS, Mehdi; DEBBAH, mérouane; POOR, H. Vincent. Ultra-Reliable and Low-Latency Wireless Communication: Tail, Risk and Scale, jan. 2018.

CAO, Liming *et al.* Cost optimization in edge computing: a survey. **Artificial Intelligence Review**, v. 57, n. 11, p. 312, 2024. ISSN 1573-7462.

CONTROL, Apache Traffic. **Traffic Control — Documentation**. Documentação oficial do Traffic Control. 2018. Disponível em: https://traffic-control-cdn.readthedocs.io/en/latest/. Acesso em: 12 out. 2025.

DOCKER, Inc. **Docker Documentation**. Repositório oficial de manuais e guias Docker. 2025. Disponível em: https://docs.docker.com/. Acesso em: 12 out. 2025.

ERGEN, Mustafa; SAOUD, Bilal; SHAYEA, Ibraheem; EL-SALEH, Ayman A.; ERGEN, Onur; INAN, Feride; TUYSUZ, Mehmet Fatih. Edge computing in future wireless networks: A comprehensive evaluation and vision for 6G and beyond. **ICT Express**, v. 10, n. 5, p. 1151–1173, 2024. ISSN 2405-9595.

ETSI. **Multi-access Edge Computing (MEC); Framework and Reference Architecture**. [*S.l.*], 2023. Disponível em: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.02.01_60/gs_mec003v030201p.pdf.

GOYAL, Piyush; GOYAL, Anurag. Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark. *In*: 2017 9th International Conference on

Computational Intelligence and Communication Networks (CICN). [*S.l.*: *s.n.*], 2017. P. 77–81.

GRAFANA LABS. **Grafana - The open observability platform**. [*S.l.*: *s.n.*], 2025. Acesso em: 25 out. 2025. Disponível em: https://grafana.com.

HANIF, Abdulelah; ILYAS, Mohammad. Enhance the Detection of DoS and Brute Force Attacks within the MQTT Environment through Feature Engineering and Employing an Ensemble Technique, ago. 2024.

HASHIM, Sarah; ENAD, Rusul; AL-KHAFAGI, Alyaa; ABDALHAMEED, Noor. The facilities of detection by using a tool of Wireshark. **Indonesian Journal of Electrical Engineering and Computer Science**, v. 31, p. 329, jul. 2023.

HMISSI, Fatma; OUNI, Sofiane. TD-MQTT: Transparent Distributed MQTT Brokers for Horizontal IoT Applications. IEEE, p. 479–486, mai. 2022.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

INFLUXDATA. **InfluxDB Documentation**. [*S.l.*], 2025. Acesso em: 3 nov. 2025. Disponível em: https://docs.influxdata.com/influxdb/v2/reference/key-concepts/design-principles/.

IOT ANALYTICS GMBH. **State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally**. [*S.l.*: *s.n.*], set. 2024. Press Release, IoT Analytics GmbH. Acessado em 04 de setembro de 2025. Disponível em: https://www.iot-analytics.com/research-blog.

LIMA, Keila; OYETOYAN, Tosin Daniel; HELDAL, Rogardt; HASSELBRING, Wilhelm. Evaluation of MQTT Bridge Architectures in a Cross-Organizational Context, 2025. arXiv: 2501.14890 [cs.SE].

MALHOTRA, Pankaj; RAMAKRISHNAN, Anusha; ANAND, Gaurangi; VIG, Lovekesh; AGARWAL, Puneet; SHROFF, Gautam. **LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection**. [*S.l.*: *s.n.*], 2016. arXiv: 1607.00148 [cs.AI]. Disponível em: https://arxiv.org/abs/1607.00148.

MAO, Yuyi; YOU, Changsheng; ZHANG, Jun; HUANG, Kaibin; LETAIEF, Khaled B. A survey on mobile edge computing: The communication perspective. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 4, p. 2322–2358, 2017.

MARWAN, Mbarek; AIT TEMGHART, Abdelkarim; OUHMI, Said; LAZAAR, Mohamed. Security, QoS and energy aware optimization of cloud-edge data centers using game theory and homomorphic encryption: Modeling and formal verification. **Results in Engineering**, v. 24, p. 102902, 2024. ISSN 2590-1230.

MUTICHIRO, Briytone; TRAN, Minh-Ngoc; KIM, Young-Han. QoS-Based Service-Time Scheduling in the IoT-Edge Cloud. **Sensors**, v. 21, n. 17, 2021. ISSN 1424-8220.

NDATINYA, Vivens; XIAO, Zhifeng; MANEPALLI, Vasudeva; MENG, Ke; XIAO, Yang. Network forensics analysis using Wireshark. **International Journal of Security and Networks**, v. 10, p. 91, jan. 2015.

SHI, Weisong; CAO, Jie; ZHANG, Quan; LI, Youhuizi; XU, Lanyu. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 5, p. 637–646, 2016.

WEN, Xianyun; LI, Weibang. Time Series Prediction Based on LSTM-Attention-LSTM Model. **IEEE Access**, v. 11, p. 48322–48331, 2023.

ZHOU, Jingwen; PAL, Shantanu; DONG, Chengzu; WANG, Kaibin. Enhancing quality of service through federated learning in edge-cloud architecture. **Ad Hoc Networks**, v. 156, p. 103430, 2024. ISSN 1570-8705.

ZHOU, Zhi; CHEN, Xu; LI, En; ZENG, Liekang; LUO, Ke; ZHANG, Junshan. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. **Proceedings of the IEEE**, PP, p. 1–25, jun. 2019.