DISSERTAÇÃO DE MESTRADO PROFISSIONAL

CAN-ESP: Uma Rede CAN de Baixo Custo e Código Aberto com Suporte a OTA para Veículos Elétricos

Danilo Moura Pereira

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CAN-ESP: A LOW-COST, OPEN-SOURCE CAN NETWORK WITH OTA SUPPORT FOR ELECTRIC VEHICLES

CAN-ESP: UMA REDE CAN DE BAIXO CUSTO E CÓDIGO ABERTO COM SUPORTE A OTA PARA VEÍCULOS ELÉTRICOS

DANILO MOURA PEREIRA

ORIENTADOR: GERALDO PEREIRA ROCHA FILHO, Ph.D

DISSERTAÇÃO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA

PPEE.MP.106
BRASÍLIA/DF, JANEIRO - 2026

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

CAN-ESP: Uma Rede CAN de Baixo Custo e Código Aberto com Suporte a OTA para Veículos Elétricos

Danilo Moura Pereira

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Geraldo Pereira Rocha Filho, Ph.D, DCET/UESB
Orientador _____

Prof. Fábio Lúcio Lopes de Mendonça, Ph.D, FT/UnB
Examinador Interno _____

Prof. José Rodrigues Torres Neto, Ph.D, DC/UFPI
Examinador Externo _____

FICHA CATALOGRÁFICA

PEREIRA, DANILO MOURA
CAN-ESP: Uma Rede CAN de Baixo Custo e Código Aberto com Suporte a OTA para Veículos Elétricos [Distrito Federal] 2026.
xvi, 81 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2026).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Redes CAN	2. Veículos Elétricos
3. Sistemas Embarcados	4. Atualização FOTA
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PEREIRA, D. M. (2026). *CAN-ESP: Uma Rede CAN de Baixo Custo e Código Aberto com Suporte a OTA para Veículos Elétricos* . Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 81 p.

Danilo Moura Pereira
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Dedico este trabalho, primeiramente, ao Deus Todo-Poderoso, fonte de toda sabedoria e fortaleza, cuja graça e amor incondicional tornaram esta conquista possível.

Aos meus filhos, Mariana, Felipe e Camila, e ao meu pequeno neto, Manoel, que são a luz dos meus dias. Que esse êxito pessoal traga inspiração para suas vidas, levando-os a enxergar o caminho para o progresso pessoal através do estudo dedicado e incessante. Vocês me inspiram, diariamente, a nunca parar de buscar novos horizontes.

À minha noiva, Cristina, pelo apoio incondicional, pelo amor e, acima de tudo, pela paciência e compreensão nos momentos de ausência e dedicação exigidos por esta jornada.

AGRADECIMENTOS

Agradeço, primeiramente, a **Deus**, que torna possível todas as coisas e que, sem Ele, nada posso fazer.

Ao meu orientador, **Professor Doutor Geraldo Pereira Rocha Filho**, exemplo de trajetória acadêmica, expresso minha mais profunda gratidão e admiração. Obrigado pela confiança depositada em mim, pela orientação segura, pelo rigor técnico exigido que elevou substancialmente a qualidade desta pesquisa e, acima de tudo, pela parceria e paciência durante todo o processo de desenvolvimento deste trabalho.

Aos meus filhos, **Mariana, Felipe e Camila**, e ao meu neto, **Manoel**. Vocês são a razão de todo o meu esforço. Obrigado por serem fontes inesgotáveis de motivação. Espero que esta conquista sirva de exemplo de que a busca pelo conhecimento não tem hora para começar e, muito menos, para acabar.

À minha noiva, **Cristina**, meu porto seguro. Obrigado por compreender minhas ausências, por me incentivar nos momentos de cansaço e por acreditar neste sonho tanto quanto eu. Sem o seu amor e apoio incondicional, o caminho teria sido muito mais árduo.

À minha **mãe**, pelo dom da vida e por todo o sacrifício dedicado à minha formação como pessoa.

À **Universidade de Brasília (UnB)** e ao **Programa de Pós-Graduação Profissional em Engenharia Elétrica (PPEE)**, pela excelência no ensino e infraestrutura que permitiram o desenvolvimento de uma pesquisa de alto nível.

À **Universidade Estadual do Sudoeste da Bahia (UESB)**, minha casa há mais de 30 anos. Agradeço por disponibilizar os recursos e o espaço para o desenvolvimento de todas as etapas práticas do projeto.

Ao **Professor Doutor Roque Trindade**, professor da UESB, pelo apoio irrestrito em todas as fases da pesquisa, sempre disponível para sanar dúvidas e auxiliar no que fosse preciso para o sucesso deste trabalho.

Ao **Professor Doutor Hélio Lopes**, professor da UESB e minha chefia no CIPEC, pela compreensão e flexibilidade em relação às minhas ausências durante todo o percurso do mestrado.

À **Professora Doutora Maísa Soares dos Santos Lopes**, professora da UESB, minha sincera gratidão pela recomendação que viabilizou meu ingresso no PPEE/UnB.

Ao colega **Bruno**, companheiro de jornada na graduação em Ciência da Computação na UESB, pelo valioso auxílio e pelas ricas discussões técnicas durante a montagem da rede CAN.

Aos **professores do PPEE**, pelos ensinamentos compartilhados em sala de aula, que foram fundamentais para a construção da base teórica que sustenta esta dissertação.

Agradeço também aos colegas de mestrado pelas discussões técnicas, sugestões e pela troca de experiências que enriqueceram esta jornada.

Por fim, a todos os amigos que, de perto ou de longe, torceram pelo meu sucesso.

RESUMO

A crescente complexidade eletrônica dos Veículos Elétricos Leves (LEVs) intensificou a diferença entre os sistemas de comunicação veicular robustos e de alto custo oferecidos pelos principais fabricantes e os protótipos acadêmicos de baixo custo que carecem de funcionalidades modernas e conformidade com os padrões da indústria. Essa diferença limita o desenvolvimento ágil e restringe a adoção de práticas de Veículos Definidos por *Software* (SDV) em ambientes de pesquisa, especialmente quando os desenvolvedores exigem controle total sobre a pilha de comunicação e os mecanismos de atualização de *firmware*.

Este trabalho apresenta o CAN-ESP, uma arquitetura de rede CAN de baixo custo e código aberto projetada para preencher essa lacuna. A arquitetura inclui um protocolo de orquestração distribuída para atualizações *Firmware-Over-The-Air* (FOTA), com um mecanismo de *feedback* de malha fechada que garante a consistência do *firmware* entre ECUs (Unidades de Controle Eletrônico) distribuídas (homogêneas em *hardware-base*, porém com funções e periféricos distintos). Uma contribuição relevante do CAN-ESP é a integração nativa do FOTA como funcionalidade da arquitetura, implementada na camada de aplicação sobre a comunicação local via ESP-WIFI-MESH e a sinalização/transferência com a nuvem via MQTT/HTTPS, permitindo atualizações confiáveis e escalonáveis sem dependência de ferramentas proprietárias.

Avaliamos o CAN-ESP em um protótipo funcional integrado a um veículo elétrico leve (LEV) e em testes de bancada. A arquitetura demonstrou capacidade de atender a requisitos de tempo real estrito, com *Worst-Case Response Time* (WCRT) de 141 μs para mensagens críticas de freio, mantendo-se abaixo de 200 μs para comandos de atuação mesmo sob carga superior a 90%. A previsibilidade temporal foi corroborada por *jitter* médio de 1,93 μs ($\sigma = 0,65$ μs) e latência de comunicação entre 129–141 μs. Quanto à robustez operacional, a rede apresentou carga do barramento entre 10%–25% em operação nominal, com taxas de retransmissão de 15 ppm (0,0015%) e de perda de arbitragem de 0,55%, sem ocorrência de estados *Bus-Off*. A combinação desses resultados demonstra que a arquitetura mantém janelas de atualização eficientes e seguras no contexto experimental investigado. Adicionalmente, a escalabilidade do processo de atualização foi analisada por modelagem teórica para cenários com dezenas de ECUs, discutindo a viabilidade de janelas de manutenção compatíveis com aplicações práticas.

Os resultados indicam que o CAN-ESP é uma plataforma funcional para pesquisa veicular, representando uma contribuição rumo à democratização do desenvolvimento de sistemas veiculares conectados e alinhados a requisitos de tempo real estrito no escopo avaliado.

ABSTRACT

The increasing electronic complexity of Light Electric Vehicles (LEVs) has intensified the gap between robust, high-cost vehicular communication systems offered by major manufacturers and low-cost academic prototypes that lack modern functionalities and compliance with industry standards. This gap limits agile development and restricts the adoption of Software-Defined Vehicle (SDV) practices in research environments, especially when developers require full control over the communication stack and the firmware update mechanisms.

This work presents CAN-ESP, a low-cost, open-source CAN network architecture designed to fill this gap. The architecture includes a distributed orchestration protocol for Firmware-Over-The-Air (FOTA) updates, with a closed-loop feedback mechanism that ensures firmware consistency across distributed ECUs (Electronic Control Units) (homogeneous in hardware-base, but with distinct functions and peripherals). A relevant contribution of CAN-ESP is the native integration of FOTA as an architectural functionality, implemented at the application layer over local communication via ESP-WIFI-MESH and signaling/transfer with the cloud via MQTT/HTTPS, enabling reliable and scalable updates without dependence on proprietary tools.

We evaluated CAN-ESP in a functional prototype integrated into a light electric vehicle (LEV) and in bench tests. The architecture demonstrated the ability to meet hard real-time requirements, with a Worst-Case Response Time (WCRT) of 141 μs for critical brake messages, remaining below 200 μs for actuation commands even under a bus load above 90%. Temporal predictability was corroborated by a mean jitter of 1.93 μs ($\sigma = 0.65$ μs) and communication latency between 129–141 μs. Regarding operational robustness, the network exhibited bus load between 10%–25% in nominal operation, with retransmission rates of 15 ppm (0.0015%) and arbitration loss rates of 0.55%, with no Bus-Off states. The combination of these results demonstrates that the architecture maintains efficient and safe update windows in the investigated experimental context. In addition, the scalability of the update process was analyzed through theoretical modeling for scenarios with dozens of ECUs, discussing the feasibility of maintenance windows compatible with practical applications.

The results indicate that CAN-ESP is a functional platform for vehicular research, representing a contribution towards democratizing the development of connected vehicular systems aligned with hard real-time requirements within the evaluated scope.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

B_i	Tempo máximo de bloqueio da mensagem i na Análise de Tempo de Resposta (RTA)
C_i	Tempo de transmissão (duração) da mensagem i no barramento
D_i	*Deadline* (prazo) associado à mensagem i
J_i	*Jitter* (variação temporal) da mensagem i
R_i	Pior Caso do Tempo de Resposta (*WCRT*) da mensagem i
S	Tamanho do *firmware* no modelo de escalabilidade OTA
T_i	Período de geração/transmissão da mensagem i
T_{total}	Tempo total estimado para propagação de uma atualização OTA
t_m	Tempo máximo de enfileiramento (*queuing delay*) na RTA
h	Número de saltos (*hops*) na rede *mesh*

Símbolos Gregos

α	Coeficiente de degradação por salto no modelo de escalabilidade
η	Rendimento efetivo (*throughput*) da rede *mesh*
η_{base}	Rendimento efetivo no primeiro salto da rede *mesh*

Siglas

ACK	*Acknowledgement*
AP	*Access Point*
BLE	*Bluetooth Low Energy*
CAN	*Controller Area Network*
CRC	*Cyclic Redundancy Check*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
CSMA/CD+AMP	*Carrier Sense Multiple Access with Collision Detection* / *Arbitration on Message Priority*
DLC	*Data Length Code*
DSP	*Digital Signal Processor*
DTC	*Diagnostic Trouble Code*
ECU	*Electronic Control Unit* (Unidade de Controle Eletrônico)
EMI	*Electromagnetic Interference*
ESP-IDF	Espressif IoT *Development Framework*
ESP32	*System-on-a-Chip* ESP32
FOTA	*Firmware Over-the-Air*
HMI	*Human–Machine Interface*
HTTPS	*Hypertext Transfer Protocol Secure*
IHM	Interface Homem–Máquina
IDS	*Intrusion Detection System*
IoT	*Internet of Things*
ISO	*International Organization for Standardization*
LEV	*Light Electric Vehicle*
MAC	*Medium Access Control* / *Message Authentication Code*
MCU	*Microcontroller Unit*
MDI	*Medium Dependent Interface*
MQTT	*Message Queuing Telemetry Transport*
OSI	*Open Systems Interconnection*
OTA	*Over-the-Air*
PMA	*Physical Medium Attachment*
PLS	*Physical Layer Signaling*
PWM	*Pulse Width Modulation*
RTA	*Response-Time Analysis*
RSSI	*Received Signal Strength Indicator*
SBC	*Single Board Computer*
SDV	*Software-Defined Vehicle*
SoC	*System-on-Chip*
SPI	*Serial Peripheral Interface*
STA	*Station mode* (Wi-Fi)

TLS	*Transport Layer Security*
TRL	*Technology Readiness Level*
TWAI	*Two-Wire Automotive Interface*
UGV	*Unmanned Ground Vehicle*
UNECE	*United Nations Economic Commission for Europe*
V2X	*Vehicle-to-Everything*
WCRT	*Worst-Case Response Time*
WLAN	*Wireless Local Area Network*
Wi-Fi	*Wireless Fidelity*

Subscritos

i	Índice da mensagem de interesse (RTA/WCRT)
j	Índice de mensagem interferente (prioridade mais alta)
$base$	Valor no primeiro salto (modelo de escalabilidade OTA)
$overhead$	Componente de sobrecarga de processamento/controle
$total$	Valor total (tempo total, somatório global)
max	Valor máximo

Sobrescritos

^{n}	Iteração n em equações de recorrência (RTA)
$^{(k)}$	k-ésima instância/transmissão de uma mensagem

1 INTRODUÇÃO

A ascensão dos veículos elétricos (EVs) tem gerado uma crescente complexidade eletrônica na indústria automotiva [5]. Para orquestrar o elevado número de Unidades de Controle Eletrônico (ECUs) que gerenciam os sistemas do veículo — desde o trem de força, passando pelos sistemas de segurança, até os dispositivos de conforto a bordo — a indústria consolidou, há mais de três décadas, o protocolo *Controller Area Network* (CAN) como a base da comunicação intraveicular [6]. Desenvolvido para garantir uma comunicação serial robusta e em tempo real [7], o protocolo CAN permanece como a tecnologia dominante não apenas no setor automotivo, mas também em áreas como automação industrial e robótica [8, 9], graças ao seu custo acessível, sua confiabilidade e ao seu sistema de arbitragem baseado em prioridade [10].

No entanto, apesar de sua adequação ao controle distribuído e a requisitos de tempo real, iniciativas de pesquisa e prototipagem frequentemente enfrentam barreiras práticas para incorporar, de forma integrada, conectividade, manutenibilidade e gestão do ciclo de vida do *software*. A Seção 1.1 situa esse problema no contexto de transformação do setor automotivo e do paradigma do Veículo Definido por *Software* (SDV) [11], delimitando a tensão existente entre a relevância persistente do CAN e a necessidade de incorporar capacidades modernas de gestão do ciclo de vida do sistema.

Nesse contexto, este trabalho de dissertação apresenta, projeta e valida a CAN-ESP: uma arquitetura de rede CAN de código aberto, alto desempenho e baixo custo, concebida para as demandas de veículos elétricos. A inovação deste trabalho vai além da escolha do *hardware*. Propomos uma máquina de estados distribuída que gerencia o ciclo de vida do *software* embarcado, mitigando o problema do 'ponto único de falha' comum em *gateways* centralizados e viabilizando a atomicidade das atualizações através de um ciclo de *feedback* estruturado. A solução não apenas garante total conformidade com o padrão ISO 11898, mas também integra de forma nativa a capacidade de atualização remota OTA.

Este trabalho contribui para o estado da arte ao apresentar uma solução sustentada em cinco pilares: baixo custo, conformidade com os padrões da indústria automotiva (ISO 11898), capacidade OTA nativa, filosofia de código aberto e validação quantitativa rigorosa. Em contraste com abordagens fragmentadas, esta dissertação detalha uma arquitetura de *hardware* integrada que engloba os elementos necessários ao seu funcionamento. Todo o sistema é gerenciado pela biblioteca `can_esp_lib`, uma API disponibilizada como código aberto para promover a sua reprodutibilidade. Por fim, apresenta-se a avaliação de desempenho realizada no UGV elétrico CELINA (Carro Elétrico Livre com Navegação Autônoma), com foco em métricas críticas de tempo real (WCRT e *jitter*) e em cenários de carga de barramento. Esta abordagem busca reduzir uma lacuna frequentemente observada em trabalhos similares, propondo um novo padrão de validação para arquiteturas de redes intraveiculares CAN.

Diante do exposto, esta dissertação é guiada pela seguinte questão de pesquisa: *É viável desenvolver e validar uma arquitetura de rede CAN para veículos elétricos que, utilizando componentes de hardware de baixo custo, atenda a requisitos de desempenho e previsibilidade temporal relevantes ao domínio automotivo e, simultaneamente, integre nativamente funcionalidades associadas ao paradigma do SDV [11], como atualizações OTA?* A resposta a essa questão será construída por meio do projeto, implementação e

validação experimental da arquitetura CAN-ESP, detalhados nos capítulos subsequentes.

1.1 CONTEXTUALIZAÇÃO

O setor automotivo tem passado por uma reconfiguração estrutural impulsionada, principalmente, por três vetores tecnológicos: a eletrificação dos sistemas de propulsão, o aumento das demandas de conectividade e a progressiva definição do veículo por *software*. Essa convergência, sintetizada no conceito de Veículo Definido por *Software* (SDV), não representa apenas uma evolução incremental, mas uma mudança no paradigma de valor e funcionalidade do automóvel, em que características antes rigidamente vinculadas ao *hardware* tornam-se passíveis de evolução contínua por meio de atualizações de *software*, preferencialmente realizadas de forma remota [11].

Essa transição exige arquiteturas elétricas e eletrônicas (E/E) mais flexíveis, seguras e adaptativas [5]. Nesse contexto, a rede de comunicação intraveicular deixa de atuar apenas como canal de dados de controle e monitoramento para assumir também o papel de plataforma de serviços, capaz de apoiar a distribuição de funcionalidades, o gerenciamento do ciclo de vida do *software* e a coexistência segura entre tráfego crítico e não crítico.

É nesse cenário de requisitos ampliados que se confrontam a herança tecnológica consolidada e a necessidade de modernização. O protocolo CAN, baseado em simplicidade, robustez e determinismo, permanece amplamente relevante como infraestrutura de comunicação para funções de controle distribuído em tempo real, particularmente em domínios sensíveis ao custo, como Veículos Elétricos Leves (LEVs) e plataformas experimentais [6, 10, 12]. Entretanto, seu ecossistema histórico não foi concebido para incorporar, de forma nativa, requisitos associados a conectividade, maior largura de banda e gestão dinâmica de *software* que se tornam centrais no paradigma SDV. Assim, emerge uma tensão arquitetural: como modernizar a rede de bordo, integrando capacidades como atualização OTA, diagnóstico remoto e interação com serviços externos, preservando a confiabilidade, o baixo custo e a ampla base instalada do CAN? A Figura 1.1 evidencia esse contraste ao comparar as taxas de transmissão de diferentes tecnologias ao longo do tempo, indicando que a evolução de largura de banda não ocorreu de forma homogênea nas aplicações veiculares e que o CAN permanece competitivo em termos de em termos de determinismo, robustez e custo, apesar de limitações para certas classes de tráfego.

Na prática, esse desalinhamento arquitetural se materializa como uma lacuna de oferta tecnológica. Soluções industriais, frequentemente baseadas em *gateways* centralizados e pilhas proprietárias, tendem a atender requisitos de desempenho e funcionalidades modernas, mas o fazem com custo elevado, baixa transparência e dependência de fornecedor, o que dificulta sua replicação em pesquisa e prototipagem ágil [13]. Por outro lado, implementações acadêmicas e experimentais de baixo custo, embora úteis como provas de conceito, não apresentam evidências claras de conformidade com requisitos industriais (ISO 11898) e tampouco mecanismos modernos de manutenção e gestão do ciclo de vida do *software*, como atualização remota *Over-The-Air* (OTA) concebida como parte constitutiva da arquitetura [12].

No recorte desta dissertação, o foco reside em redes intraveiculares aplicadas a veículos elétricos leves e plataformas experimentais, nas quais múltiplas ECUs executam diversas funções e exigem comunicação

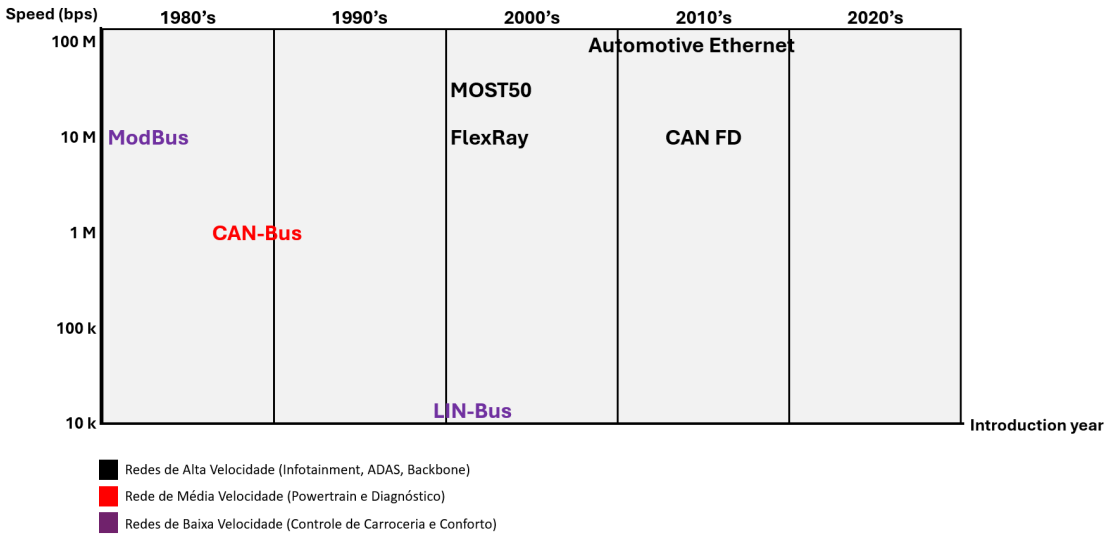

Figura 1.1: Panorama da evolução histórica das redes de comunicação intraveicular. Fonte: elaborada pelo autor (2025).

confiável sob requisitos de tempo real estrito. Esse cenário motiva a investigação de arquiteturas que preservem o determinismo do CAN e, simultaneamente, incorporem conectividade e mecanismos de gestão do ciclo de vida do *software* em uma plataforma de baixo custo, mantendo aderência ao padrão ISO 11898 e sustentadas por validação experimental rigorosa.

1.2 JUSTIFICATIVA

A relevância deste trabalho se fundamenta em três pilares interdependentes: (i) a necessidade de evolução tecnológica nas redes veiculares, especialmente no que se refere à conectividade, à gestão remota do *software* embarcado e à validação de requisitos de tempo real; (ii) a existência de uma clara lacuna na literatura acadêmica; e (iii) o potencial de impacto para a inovação no setor de mobilidade elétrica.

A justificativa tecnológica emerge da crescente complexidade das arquiteturas elétricas e eletrônicas (E/E) em veículos elétricos [5], que atualmente demandam redes de comunicação não apenas robustas, mas também flexíveis e adaptáveis, especialmente em plataformas experimentais e Veículos Elétricos Leves (LEVs), onde restrições de custo e ciclos rápidos de prototipagem são determinantes. O protocolo CAN, embora consolidado por sua confiabilidade e baixo custo, enfrenta limitações em implementações tradicionais quando se busca incorporar, de forma integrada, funcionalidades modernas de conectividade e manutenibilidade, como atualizações remotas de *firmware*. Essa limitação não é apenas um inconveniente: no contexto do Veículo Definido por *Software* (SDV), ela se torna um gargalo de cibersegurança e manutenção, pois dificulta a correção ágil de vulnerabilidades e a evolução de funcionalidades ao longo do ciclo de vida do sistema [11]. Este trabalho, portanto, se justifica por propor uma arquitetura que combina a base determinística e a confiabilidade do protocolo CAN com a flexibilidade e a conectividade nativa de um *System-on-Chip* (SoC) moderno, integrando a capacidade de atualização remota (OTA) como elemento constitutivo da solução e viabilizando a evolução do *software* embarcado sem intervenções físicas

recorrentes, um requisito fundamental para a manutenibilidade dos sistemas automotivos atuais.

Do ponto de vista acadêmico, a presente dissertação se justifica por endereçar uma lacuna multidimensional na literatura especializada. A revisão da área, detalhada no Capítulo 3, indica que, no conjunto de trabalhos analisados, não foi identificada evidência de uma solução que integre, de forma coesa e validada, os cinco critérios de comparação adotados neste trabalho: (1) baixo custo, possibilitando replicação em larga escala; (2) conformidade com os padrões da indústria automotiva (ISO 11898); (3) capacidade nativa de atualização remota OTA; (4) arquitetura de *software* de código aberto; e (5) avaliação de desempenho quantitativa rigorosa. Os estudos existentes tipicamente sacrificam conformidade e funcionalidades em prol do baixo custo, como em abordagens baseadas em Arduino [14], ou dependem de *hardware* proprietário e de maior custo, como a plataforma de alto desempenho de Coelho e Silva-Filho (2024) [15]. Além disso, a análise conduzida nesta dissertação avança ao investigar não apenas carga de barramento, mas também garantias de tempo real, como o Pior Caso do Tempo de Resposta (WCRT) e o *jitter* de mensagens, aspectos essenciais para segurança funcional e ainda frequentemente pouco explorados em validações experimentais de baixo custo.

Finalmente, o trabalho se justifica pela sua relevância prática e potencial de contribuição para além da academia. Ao oferecer uma plataforma de desenvolvimento de rede veicular que é, ao mesmo tempo, poderosa, de baixo custo e aberta, o CAN-ESP contribui para a democratização da inovação no setor. A solução proposta tem o potencial de auxiliar na redução da barreira de entrada para startups, laboratórios de pesquisa universitários e equipes de competição estudantil (e.g., Fórmula SAE), que projetam sistemas veiculares elétricos complexos [16]. Dessa forma, ao disponibilizar esta tecnologia, este trabalho não apenas avança o conhecimento na área, mas também pode servir como uma base sólida para o desenvolvimento futuro de aplicações em mobilidade inteligente, segurança veicular funcional e infraestrutura conectada, fomentando um ecossistema de inovação ágil e acessível.

1.3 OBJETIVOS

Para responder à questão de pesquisa apresentada, este trabalho tem como objetivo geral investigar, propor e validar uma arquitetura de rede CAN de baixo custo e código aberto para veículos elétricos, considerando aspectos de robustez, desempenho em tempo real e conformidade com padrões da indústria automotiva, com suporte nativo a atualizações remotas de *firmware* (OTA).

Neste contexto, o termo robustez é empregado em uma abordagem multidimensional, caracterizando a arquitetura CAN-ESP em quatro aspectos fundamentais que vão além da estabilidade operacional. Em primeiro lugar, a robustez protocolar refere-se à aderência às especificações da norma ISO 11898, visando garantir que a sinalização diferencial e a lógica de arbitragem bit a bit sejam preservadas, o que busca assegurar que a utilização de componentes de baixo custo não comprometa a interoperabilidade e a integridade que são requisitos do padrão CAN. A robustez temporal, por sua vez, está relacionada à capacidade do sistema em manter tempos de resposta previsíveis e dentro de limites especificados (*deadlines*), mesmo sob condições de alta carga do barramento (superior a 90%), buscando assegurar o determinismo necessário para funções de controle críticas. Essa dimensão é avaliada por meio da análise do Pior Caso de

Tempo de Resposta (*Worst-Case Response Time* – WCRT), que mede a capacidade do barramento de garantir que mensagens de alta prioridade cumpram seus *deadlines* de comunicação mesmo em cenários de saturação do barramento (superior a 90%). Em terceiro lugar, a robustez operacional descreve a resiliência da arquitetura distribuída frente a falhas individuais de componentes, mantendo a operação funcional do sistema, que é monitorada por meio do mecanismo de confinamento de erro (*error confinement*). Essa dimensão visa garantir que as ECUs operem predominantemente em estado normal (*Error Active*) e que a taxa de retransmissão de quadros se mantenha em patamares mínimos, da ordem de partes por milhão. Por fim, a robustez de comunicação é assegurada pelos mecanismos intrínsecos do protocolo CAN, incluindo detecção e correção de erros, confirmação de recepção e retransmissão automática, complementados por uma implementação adequada da camada física que busca garantir a integridade dos dados mesmo em ambientes sujeitos a interferência eletromagnética (*Electromagnetic Interference* – EMI).

Para alcançar este propósito, foram definidos os seguintes objetivos específicos:

1) **Projetar** uma arquitetura de rede intraveicular aderente à norma ISO 11898, utilizando componentes de baixo custo e *software* de código aberto, e que integre nativamente funcionalidades de atualização remota de *firmware* (OTA).

2) **Desenvolver** um protótipo físico da arquitetura, composto por cinco ECUs homogêneas baseadas no SoC ESP32, e implementar a biblioteca de comunicação `can_esp_lib` para orquestrar o fluxo de dados e o gerenciamento da rede.

3) **Validar** o determinismo e a robustez temporal da arquitetura através da análise experimental de métricas de tempo real, buscando garantir a previsibilidade necessária para sistemas de segurança funcional.

4) **Validar** a aplicabilidade e a robustez operacional e de comunicação da arquitetura CAN-ESP através de sua integração e teste em um veículo elétrico real (UGV CELINA – Carro Elétrico Livre com Navegação Autônoma), demonstrando seu funcionamento em um cenário operacional.

1.4 RESULTADOS ALCANÇADOS

Os principais resultados obtidos indicam a viabilidade técnica da arquitetura CAN-ESP. Através de testes em bancada e no veículo UGV CELINA, observou-se que o sistema preserva a integridade da comunicação e opera em conformidade com o comportamento esperado para uma rede baseada em ISO 11898 nos cenários de carga avaliados. Ademais, a instrumentação de *software* permitiu a caracterização do Pior Caso de Tempo de Resposta (WCRT) e do *jitter*, contribuindo para a discussão de adequação da arquitetura a requisitos de tempo real estrito. O mecanismo de atualização OTA distribuído apresentou comportamento resiliente, permitindo atualizações coordenadas de *firmware* em múltiplas ECUs com monitoramento por estados e *feedback*, sem comprometer a estabilidade operacional durante os ensaios reportados.

1.5 PUBLICAÇÕES RESULTANTES DESTA PESQUISA

1.5.1 Trabalho Publicado

Os resultados da avaliação de desempenho em bancada e a arquitetura central desenvolvidos nesta dissertação foram consolidados e aceitos para publicação no **IX Workshop de Computação Urbana (CoUrb 2025)**, evento integrante do 43º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2025), conforme referência bibliográfica a seguir:

- PEREIRA, Danilo Moura; RODRIGUES-FILHO, Roberto; GONÇALVES, Vinícius P.; DOS SANTOS, Hélio L.; TRINDADE, Roque M. P.; MENEGUETTE, Rodolfo; SERRANO, André L. Marques; ROCHA FILHO, Geraldo P.. CAN-ESP: Rede CAN de Baixo Custo para Veículos Elétricos. *In*: WORKSHOP DE COMPUTAÇÃO URBANA (COURB), 9., 2025, Natal/RN. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2025. p. 1-14. ISSN 2595-2706. DOI: <https://doi.org/10.5753/courb.2025.7060>. [17]

1.5.2 Trabalho Submetido

Visando a divulgação internacional dos resultados completos e da avaliação em protótipo integrado a um veículo, um artigo expandindo significativamente o trabalho apresentado no Workshop de Computação Urbana (COURB 2025) foi submetido a um periódico internacional de alto fator de impacto:

- PEREIRA, Danilo Moura; RODRIGUES-FILHO, Roberto; GONÇALVES, Vinícius P.; DOS SANTOS, Hélio L.; TRINDADE, Roque M. P.; MENEGUETTE, Rodolfo; SERRANO, André L. Marques; ROCHA FILHO, Geraldo P.. CAN-ESP: A low-cost, open-source Controller Area Network architecture with OTA support for electric vehicles. *Future Generation Computer Systems*. Elsevier. (Submetido em: 09 de dezembro de 2025. Status: *Under Review*. Manuscript number: FGCS-D-25-04816).

1.6 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está dividida em seis capítulos. O Capítulo 1 apresenta o contexto, a motivação, a questão de pesquisa, os objetivos e uma síntese dos resultados e contribuições. O Capítulo 2 estabelece o alicerce teórico, abordando os princípios do protocolo CAN, seu funcionamento e os conceitos essenciais para a compreensão do projeto. O Capítulo 3 realiza uma análise crítica e sistemática da literatura, identificando lacunas em custo, conectividade e validação que justificam a necessidade da arquitetura proposta. O Capítulo 4 detalha a concepção e a estrutura da arquitetura CAN-ESP, abrangendo sua filosofia de *design*, os componentes de *hardware* e de *software*, o modelo de comunicação e o mecanismo de atualização OTA. O Capítulo 5 é dedicado à validação experimental e à análise de desempenho, apresentando a metodologia e os resultados da implementação e teste da rede no UGV CELINA. Por fim, o Capítulo 6 consolida os resultados da pesquisa, resume as contribuições e aponta direções para investigações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo estabelece o alicerce teórico necessário para a adequada compreensão da arquitetura CAN-ESP e suas contribuições. A discussão inicia com a contextualização do protocolo CAN dentro do modelo de referência OSI, estabelecendo seu escopo e filosofia de projeto. Em seguida, são explorados os princípios fundamentais de sua arquitetura, com uma ênfase especial na sua adequação para sistemas de tempo real, introduzindo a Análise de Tempo de Resposta (RTA) como ferramenta de validação, detalhando as métricas de tempo real estrito, como WCRT e *jitter*. A análise prossegue para a camada física, detalhando não apenas os componentes de *hardware* essenciais, mas também as limitações práticas de topologia e comprimento de barramento. Em seguida, o capítulo disseca a estrutura do *frame* de dados de uma rede CAN, descrevendo sua anatomia e o processo de recepção e filtragem das mensagens dentro do barramento. Finalmente, para prover o embasamento da funcionalidade mais inovadora do sistema, o capítulo introduz a arquitetura das redes *mesh* sem fio, detalhando a tecnologia que permite a atualização remota de *firmware*. Ao final deste capítulo, o leitor possuirá todo o ferramental teórico para compreender as decisões de *design* e as contribuições da arquitetura CAN-ESP.

2.1 O PROTOCOLO CAN

2.1.1 CAN e o Modelo de Referência OSI

Para contextualizar o *Controller Area Network* (CAN) no panorama mais amplo dos protocolos de comunicação, é útil analisá-lo sob a ótica do Modelo de Referência de Interconexão de Sistemas Abertos (ISO/OSI). Embora o modelo OSI de sete camadas seja uma referência abrangente, o CAN foi especificado de forma intencionalmente focada e eficiente: a família ISO 11898 define a Camada de Enlace de Dados (*Data-Link Layer*) (ISO 11898-1) e variantes da Camada Física (*Physical Layer*) (por exemplo, ISO 11898-2 e ISO 11898-3), restringindo-se, portanto, às duas camadas inferiores do modelo OSI (*Layer* 1 e *Layer* 2) e deixando as camadas superiores fora do escopo do protocolo base [18, 12].

A camada física do CAN é responsável por todos os aspectos de *hardware* da transmissão, incluindo a conversão dos bits em sinais elétricos, a definição dos níveis de tensão e a temporização dos bits no barramento [19]. Conforme ilustrado na Figura 2.1, esta camada é frequentemente subdividida em três partes funcionais: *Physical Signaling* (PLS), *Physical Medium Attachment* (PMA), e *Medium Dependent Interface* (MDI) [19]. O diagrama mostra a correspondência entre as camadas OSI, as subcamadas teóricas do protocolo CAN (conforme ISO 11898) e os componentes de *hardware* e *software* que as implementam neste projeto.

A camada de enlace de dados, por sua vez, é responsável por garantir que os dados sejam transferidos de forma confiável através do barramento. Suas funções incluem o enquadramento das mensagens (*framing*), o mecanismo de arbitragem para acesso ao meio, a detecção de erros (e.g., CRC) e os mecanismos de confirmação (*acknowledgement*) [7]. Esta camada também é subdividida em duas subcamadas: *Medium*

Access Control (MAC), que gerencia o acesso ao barramento e a arbitragem, e *Logic Link Control* (LLC), que lida com a filtragem de mensagens e a notificação de sobrecarga [19].

É importante notar que as camadas superiores do modelo OSI — Rede (*Layer* 3), Transporte (*Layer* 4), Sessão (*Layer* 5), Apresentação (*Layer* 6) e Aplicação (*Layer* 7) — estão intencionalmente fora do escopo da especificação base do CAN (ISO 11898). Essa omissão é uma decisão de projeto fundamental que busca garantir a baixa latência e a alta eficiência do protocolo, eliminando o *overhead* associado a funções como roteamento de pacotes entre redes distintas ou gerenciamento de sessões de comunicação. Isso ocorre porque o CAN foi concebido não como uma rede de dados de propósito geral, mas como um barramento de controle em tempo real, onde conceitos como endereçamento de rede (*Layer* 3) ou conexões ponto a ponto (*Layer* 4) são desnecessários e apenas introduziriam latência adicional.

As funcionalidades da Camada de Aplicação (*Application Layer*), que definem como interpretar os dados, gerenciar a rede e o comportamento dos dispositivos, são deixadas a cargo de protocolos de nível superior ou da implementação do próprio projetista [12]. No contexto deste trabalho, esta função é assumida pela biblioteca `can_esp_lib`, que opera sobre o controlador TWAI para implementar a lógica específica da arquitetura CAN-ESP. Protocolos como o CANopen [20], por exemplo, foram desenvolvidos especificamente para padronizar esta camada sobre a base robusta fornecida pelo CAN, definindo um dicionário de objetos, serviços de comunicação e gerenciamento de rede. Esta abordagem em camadas permite que o CAN mantenha sua eficiência para controle em tempo real, enquanto oferece a flexibilidade para a construção de sistemas de comunicação complexos e padronizados.

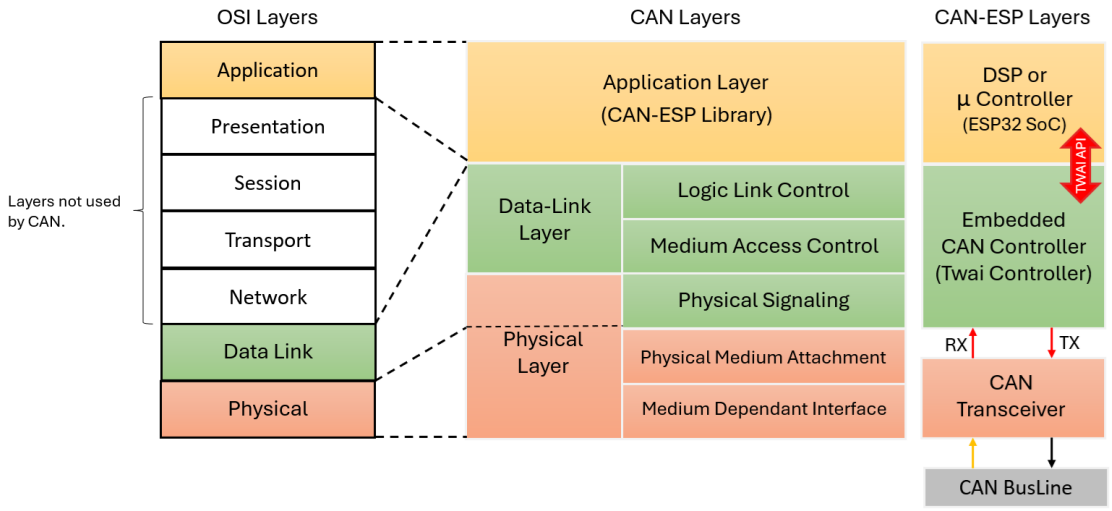

Figura 2.1: Mapeamento da arquitetura CAN-ESP em relação ao modelo de referência OSI. Fonte: elaborada pelo autor (2025).

2.1.2 CAN: Princípios e Arquitetura

O *Controller Area Network* (CAN) é um protocolo de comunicação serial multimestre, concebido na década de 1980 pela Robert Bosch GmbH para responder a uma necessidade crítica da crescente eletrônica embarcada nos veículos: substituir as complexas e pesadas fiações ponto a ponto por um único barramento de par trançado que fosse robusto, confiável e de baixo custo [6]. O sucesso de sua arquitetura o tornou

o padrão *de facto* na indústria automotiva e permitiu sua ampla adoção em diversas outras áreas, como automação industrial, sistemas aviônicos e equipamentos médicos [21].

A longevidade e difusão do CAN se devem a um conjunto de características intrínsecas, definidas na especificação original do protocolo [7], que garantem um desempenho determinístico e seguro. Estes princípios fundamentais podem ser agrupados em dois eixos estratégicos.

O primeiro eixo é a robustez e a confiabilidade. A integridade da comunicação é o pilar do protocolo CAN. Ele possui sofisticados mecanismos de detecção de erros em múltiplas camadas, incluindo verificação por redundância cíclica (CRC), monitoramento de bits em nível físico e checagem de formato de *frame*. Essencialmente, qualquer nó na rede pode detectar e sinalizar um erro, levando ao descarte imediato da mensagem corrompida por todos os nós da rede. Adicionalmente, o protocolo implementa um sistema de confinamento de falhas (*fault confinement*), um mecanismo de autodiagnóstico que permite a um nó se desligar autonomamente do barramento caso esteja gerando erros persistentemente, protegendo assim a integridade de toda a comunicação [6, 21]. Esta robustez intrínseca ao protocolo é o que permite a construção de sistemas confiáveis mesmo com *hardware* de baixo custo, como o proposto neste trabalho, pois parte da responsabilidade pela integridade da comunicação é delegada ao próprio *hardware* do controlador CAN.

O segundo eixo estratégico é o gerenciamento de barramento e o tempo real. O acesso ao meio é gerenciado por um método de arbitragem não destrutivo e baseado em prioridade, frequentemente descrito na literatura como um esquema do tipo *Carrier Sense Multiple Access* with *collision resolution* (Acesso Múltiplo com Detecção de Portadora com Resolução de Colisão por arbitragem – CSMA/CR), isto é, múltiplos nós podem iniciar transmissão após detectar o meio livre, e uma contenção é resolvida de forma não destrutiva pela prioridade da mensagem [7, 12]. Diferentemente da Ethernet, em que colisões corrompem quadros e exigem retransmissão, no CAN, se dois ou mais nós tentam transmitir simultaneamente, ocorre uma arbitragem bit a bit no identificador (ID) da mensagem. O nó que transmite a mensagem de maior prioridade (menor valor numérico de ID) vence a arbitragem e continua a transmissão sem perda de tempo ou de dados, enquanto os demais nós interrompem a transmissão e aguardam uma nova oportunidade. É este comportamento determinístico que garante que, em um cenário veicular, um comando de frenagem (alta prioridade) não será atrasado por uma mensagem de status de comunicação (baixa prioridade), um princípio fundamental para a segurança funcional. Este mecanismo de arbitragem por prioridade é o pilar sobre o qual todo o esquema de identificação de mensagens do CAN-ESP, a ser detalhado no Capítulo 4, é construído para garantir que comandos críticos sempre tenham precedência no barramento.

Estes princípios são formalizados na família de padrões ISO 11898, que especifica desde a camada de enlace de dados (ISO 11898-1) até a camada física para diferentes velocidades e modos de operação (ISO 11898-2, ISO 11898-3), garantindo a interoperabilidade que é crucial no setor automotivo [18].

2.1.3 Análise de Tempo Real em Redes CAN

A aplicação bem-sucedida do protocolo CAN em ambientes críticos, como o automotivo, depende não apenas de sua robustez funcional, mas de sua capacidade de operar como a espinha dorsal de um sistema de tempo real. Um sistema de tempo real é aquele cujo correto funcionamento depende não somente do

resultado lógico das computações, mas também do instante em que esses resultados são produzidos [22]. Em arquiteturas automotivas voltadas à segurança funcional, funções como frenagem, direção e gerenciamento do trem de força são tipicamente tratadas sob requisitos temporais estritos, nos quais a violação de prazos (*deadlines*) pode levar a consequências severas [23]. No contexto desta dissertação, ainda que o protótipo empregue *hardware* COTS (*Commercial Off-The-Shelf*) e não tenha como objetivo a certificação automotiva, a análise adota o enquadramento de tempo real estrito como referência de engenharia para discutir limites superiores de atraso e previsibilidade temporal no barramento CAN. Portanto, não basta que uma mensagem de frenagem seja entregue corretamente; ela precisa ser entregue dentro de um intervalo de tempo máximo e predeterminado, sob quaisquer condições operacionais do barramento.

A adequação do CAN para sistemas de tempo real estrito reside em seu mecanismo de arbitragem determinístico. Como explicado na seção anterior, o acesso ao barramento baseado em prioridades garante que, em caso de contenção, a mensagem mais crítica (com o menor ID) sempre vencerá a disputa, sem perda de tempo. Essa previsibilidade é a base que permite a aplicação de metodologias formais de verificação, como a Análise de Tempo de Resposta (*Response-Time Analysis* – RTA) [23, 22]. A RTA é um conjunto de técnicas matemáticas utilizadas para calcular o limite superior do tempo que cada mensagem levará para ser transmitida com sucesso, desde o momento em que é solicitada. Esse limite é conhecido como Pior Caso do Tempo de Resposta (*Worst-Case Response Time* – WCRT).

Para compreender a RTA e, consequentemente, a avaliação de desempenho que será realizada nesta dissertação, é crucial definir alguns conceitos-chave. Para uma dada mensagem *i* na rede, temos:

- **Tempo de Transmissão (C_i):** É o tempo necessário para transmitir fisicamente todos os bits da mensagem *i*, incluindo cabeçalhos, dados, CRC e bits de *stuffing*, na velocidade nominal do barramento.
- ***Deadline* (D_i):** Representa o prazo máximo, contado a partir do instante em que a mensagem *i* é enfileirada para transmissão, dentro do qual ela deve ser concluída com sucesso para que o sistema funcione corretamente.
- **Período (T_i):** Para mensagens periódicas (e.g., leitura de um sensor de velocidade), é o intervalo de tempo fixo entre duas liberações consecutivas da mensagem *i*.
- **Pior Caso do Tempo de Resposta (WCRT, R_i):** É o maior tempo possível que a mensagem *i* pode levar para ser transmitida com sucesso após ser liberada. O objetivo da análise é garantir que, para todas as mensagens críticas *i*, a condição $R_i \leq D_i$ seja sempre satisfeita.
- ***Jitter*:** É a variação no tempo de resposta para uma mesma mensagem periódica. Um *jitter* elevado pode desestabilizar malhas de controle que dependem de informações temporalmente consistentes.

A Figura 2.2 ilustra graficamente como esses componentes temporais interagem para formar o Pior Caso do Tempo de Resposta de uma mensagem de interesse. O diagrama mostra o cenário em que a mensagem sofre o atraso máximo possível devido ao bloqueio por uma mensagem de menor prioridade e à interferência por múltiplas mensagens de maior prioridade. O tempo de resposta R_i é a soma de três componentes: (1) o tempo de bloqueio B_i causado por uma mensagem de menor prioridade m_k que já estava em transmissão; (2) a interferência causada por múltiplas transmissões de mensagens de maior

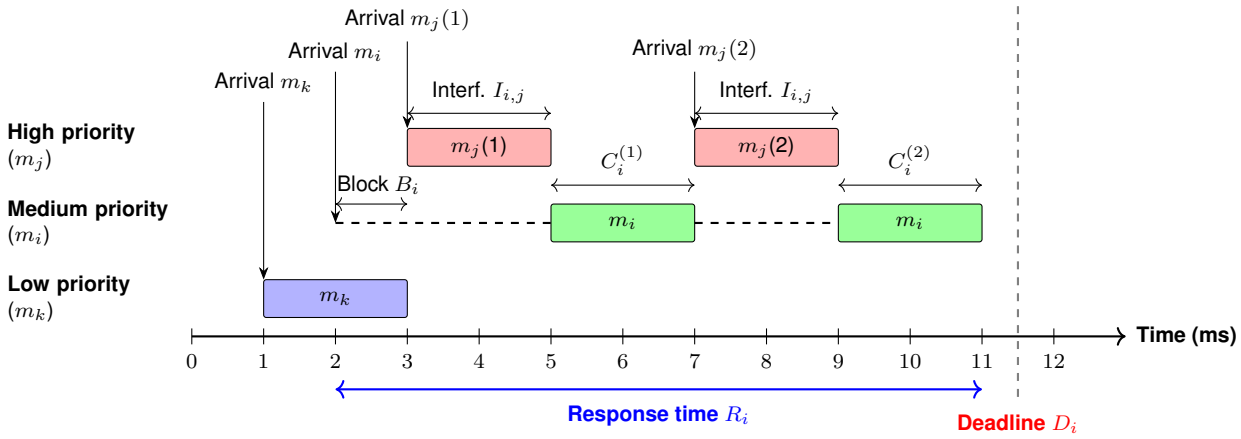

Figura 2.2: Ilustração dos componentes do Pior Caso do Tempo de Resposta (WCRT) para uma mensagem de interesse m_i. Fonte: elaborada pelo autor (2025).

prioridade $m_j(1)$ e $m_j(2)$; e (3) o tempo de transmissão da própria mensagem, C_i. Para que o sistema seja considerado válido, o WCRT (R_i) deve ser menor ou igual ao seu *deadline* (D_i).

A base matemática para a RTA em redes CAN foi estabelecida em trabalhos seminais, que propõem um método para calcular o WCRT em duas etapas [24]. Primeiro, calcula-se o tempo máximo de enfileiramento (t_m), que representa o maior tempo que uma mensagem *m* pode ser atrasada por bloqueio e por interferência de mensagens mais críticas. Este tempo é encontrado resolvendo-se a seguinte equação de recorrência:

$$t_m^{n+1} = B_m + \sum_{j \in hp(m)} \left\lceil \frac{t_m^n + J_j}{T_j} \right\rceil C_j \tag{2.1}$$

Nesta equação, a iteração busca o valor de t_m; B_m é o tempo de bloqueio máximo que *m* pode sofrer por uma única mensagem de menor prioridade; e o somatório representa a interferência total causada por todas as mensagens *j* com prioridade mais alta que *m* ($j \in hp(m)$), onde J_j representa o *jitter* da mensagem interferente j, que pode aumentar o número de instâncias interferentes no intervalo de interesse. Uma vez que o pior caso de tempo de enfileiramento (t_m) é encontrado (quando $t_m^{n+1} = t_m^n$), o Pior Caso do Tempo de Resposta (WCRT) total da mensagem, R_m, é a soma do seu tempo de enfileiramento com seu próprio tempo de transmissão (C_m):

$$R_m = t_m + C_m \tag{2.2}$$

Este método permite verificar formalmente a condição fundamental de atendibilidade temporal, isto é, se $R_m \leq D_m$ para as mensagens críticas.

Embora o cálculo analítico exaustivo de todos os tempos de resposta (*schedulability analysis*) possa ser complexo em sistemas dinâmicos, compreender seus princípios é fundamental, pois eles estabelecem o arcabouço teórico para a validação de sistemas CAN críticos. A abordagem deste trabalho, portanto, complementa a teoria clássica através de uma validação empírica rigorosa. A avaliação de desempenho apresentada no Capítulo 5 utiliza a instrumentação integrada na `can_esp_lib` para medir experimentalmente o tempo de resposta e o *jitter* na rede CAN-ESP. Esta metodologia oferece uma evidência prática

robusta da capacidade da arquitetura em atender a requisitos de tempo real estrito sob condições operacionais reais em um protótipo, validando o uso de *hardware* de baixo custo em aplicações de segurança funcional nesse cenário.

2.2 A CAMADA FÍSICA: DA TEORIA À IMPLEMENTAÇÃO

Enquanto o protocolo CAN define as regras lógicas da comunicação, é na camada física que os bits se transformam em sinais elétricos capazes de trafegar pela rede. A robustez do CAN depende fundamentalmente da sua implementação física, que é baseada no princípio da sinalização diferencial. Em vez de um único fio com tensão referenciada ao terra, o barramento CAN utiliza um par de fios trançados, denominados CAN High (CAN_H) e CAN Low (CAN_L). A informação não é representada por um nível de tensão absoluto, mas pela diferença de tensão entre esses dois fios. Este método confere uma elevada imunidade a ruídos eletromagnéticos, como os gerados por motores e inversores em um VE [21]. Isso é particularmente crítico em EVs devido às altas correntes e à comutação em alta frequência dos inversores de potência, que são fontes significativas de interferência eletromagnética (EMI). A geração e interpretação desses sinais em um nó da rede envolvem a interação de três elementos de *hardware* distintos (camada de aplicação, o controlador TWAI integrado ao ESP32 e o transceptor, que faz a interface com o barramento), conforme ilustrado na Figura 2.3.

O primeiro ator é o microcontrolador (MCU), a unidade central de processamento (o "cérebro" do nó), onde a lógica da aplicação é executada. É ele quem decide qual dado enviar ou como reagir a um dado recebido. Na arquitetura CAN-ESP, esta função é desempenhada pelo SoC ESP32.

O segundo ator é o controlador CAN, o *hardware* especialista que implementa a lógica do protocolo CAN, lidando com o enquadramento de mensagens, a arbitragem, a sinalização de erros e a filtragem de mensagens recebidas. Ele alivia o MCU principal dessas tarefas de baixo nível. No ESP32, o controlador é um periférico integrado que suporta a especificação CAN 2.0B, oficialmente denominado pela fabricante como TWAI (*Two-Wire Automotive Interface*) [25]. Essa integração entre MCU e controlador em um único chip é uma vantagem fundamental da arquitetura CAN-ESP, pois reduz drasticamente o custo, a complexidade e a área ocupada na placa de circuito impresso em comparação com sistemas que utilizam componentes discretos.

O terceiro e último ator é o Transceptor CAN (*transceiver*), que atua como a interface entre o mundo digital do controlador e o mundo analógico do barramento (transmissão física). Ele converte os sinais lógicos de transmissão (TX) do controlador nos sinais elétricos diferenciais (CAN_H/CAN_L) e, no sentido inverso, converte os sinais do barramento de volta para um sinal lógico de recepção (RX) para o controlador. Ele é, essencialmente, o "músculo" elétrico do nó [19]. O projeto CAN-ESP utiliza o transceptor SN65HVD230 para esta função.

O resultado prático dessa interação pode ser visualizado em um osciloscópio, como mostra a Figura 2.4. Nela, observa-se claramente os dois níveis de tensão distintos: no bit dominante (lógico '0'), há uma diferença de tensão significativa entre CAN_H (aprox. 3.5V) e CAN_L (aprox. 1.5V). No bit recessivo (lógico '1'), ambas as linhas tendem a um nível de tensão comum (aprox. 2.5V), resultando em uma

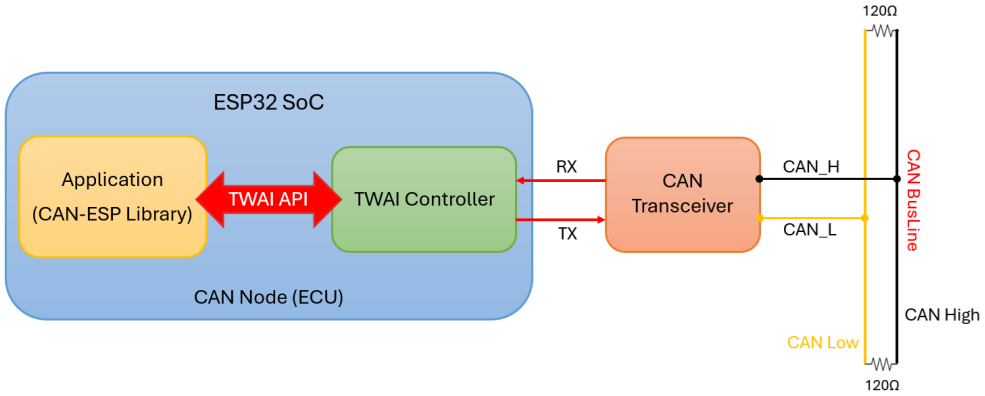

Figura 2.3: Componentes de um nó da rede CAN-ESP. Fonte: elaborada pelo autor (2025).

diferença de tensão próxima de zero.

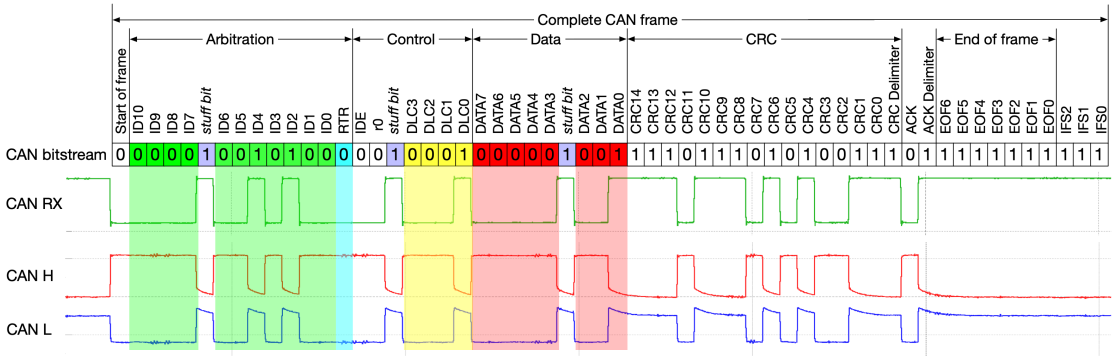

Figura 2.4: Sinais elétricos em uma transmissão no barramento CAN, ilustrando os níveis de tensão diferenciais para bits dominantes e recessivos [1]. Fonte: Wikipedia (2025) [2].

Finalmente, para garantir a integridade desses sinais ao longo do barramento, a norma ISO 11898-2 para CAN de alta velocidade (até 1 Mbps) exige que as duas extremidades da rede sejam terminadas com resistores de 120 Ω. Esses resistores de terminação são críticos para impedir a reflexão do sinal, que poderia causar interferência e corromper as mensagens, garantindo uma comunicação limpa e confiável em todo o seu comprimento [18].

2.2.1 Topologia do Barramento e Limitações Físicas

A norma ISO 11898 especifica que a topologia para redes CAN de alta velocidade (até 1 Mbps) deve ser um **barramento linear único**, terminado em ambas as extremidades com resistores de 120 Ω, conforme já mencionado. Esta topologia é essencial para garantir a integridade do sinal, pois minimiza as reflexões que poderiam ocorrer em topologias mais complexas, como estrela ou anel, que não são padronizadas para operação em alta velocidade. A escolha e o respeito a esta topologia são o primeiro passo para garantir uma comunicação confiável.

Contudo, a principal limitação de engenharia em uma rede CAN é o *trade-off* intrínseco entre a taxa de transmissão e o comprimento máximo do barramento. A razão para esta interdependência reside no tempo de propagação do sinal. Para que o mecanismo de arbitragem funcione corretamente, um bit precisa

ter tempo suficiente para se propagar até a extremidade mais distante do barramento e retornar ao nó transmissor antes que o ponto de amostragem daquele bit ocorra. Isso garante que um nó, ao transmitir um bit recessivo, consiga detectar a tempo se outro nó está transmitindo um bit dominante, perdendo assim a arbitragem. Se o barramento for demasiadamente longo para uma dada taxa de bits, a arbitragem pode falhar, levando à corrupção de *frames* e erros na rede [20]. A Tabela 2.1 apresenta os comprimentos de barramento máximos recomendados para taxas de transmissão padrão.

Tabela 2.1: Relação entre Taxa de Transmissão e Comprimento Máximo do Barramento CAN [4].

Taxa de Transmissão	**Comprimento Máximo do Barramento**
1 Mbps	40 m
500 kbit/s	100 m
250 kbit/s	200 m
100 kbit/s	500 m
50 kbit/s	1.000 m

Outra limitação física crítica na implementação da rede são as derivações (*stubs*), que são os segmentos de cabo que conectam uma ECU ao tronco principal do barramento. *Stubs* longos agem como linhas de transmissão não terminadas, criando um desequilíbrio de impedância que causa reflexões de sinal e degrada a qualidade da comunicação em toda a rede. Para redes de alta velocidade, a regra de engenharia é manter os *stubs* o mais curtos possível. Guias práticos e notas de aplicação recomendam comprimentos máximos bastante restritos; para uma rede de 1 Mbit/s, valores na ordem de decímetros (por exemplo, $\approx 0{,}3$ m) são frequentemente adotados como referência de projeto [4]. Este foi um critério de *design* fundamental na montagem do protótipo CAN-ESP, onde o comprimento do barramento e das derivações foi mantido bem abaixo desses limites para maximizar a integridade do sinal durante os testes. O desrespeito a essa limitação é uma das causas mais comuns de erros intermitentes e de difícil diagnóstico em redes CAN. Quanto ao número de nós, limites como "$\sim$30 nós" devem ser interpretados como valores típicos sob determinadas premissas de carga elétrica do barramento (por exemplo, transceptores de *unit load* e orçamento de capacitância [1]), e não como um limite universal do protocolo; em cenários mais exigentes, pode ser necessário empregar transceptores de menor carga (*fractional unit load*), revisar o orçamento de capacitância, ajustar temporizações de bit e, quando aplicável, adotar topologias segmentadas ou repetidores/isoladores para preservar margens de integridade de sinal [4].

Portanto, a implementação bem-sucedida da camada física não se resume à escolha correta dos componentes, mas exige um projeto cuidadoso do layout físico da rede. O comprimento total do barramento e o comprimento de cada derivação devem ser estritamente controlados em função da taxa de transmissão desejada, garantindo a robustez e a confiabilidade que são as principais características do protocolo CAN.

[1] O orçamento de capacitância (*capacitance budget*) na rede CAN refere-se ao limite máximo de capacitância total acumulada que o barramento (gerada por cabos longos, qualidade dos conectores e número de nós) pode suportar para garantir a integridade do sinal e a comunicação confiável. Como a rede CAN utiliza um par trançado para transmitir sinais de alta velocidade, uma capacitância muito alta distorce o formato da onda causando erros de comunicação. Valores de referência frequentemente citados: em torno de um máximo de 35 a 70 pF/m em redes de alta velocidade.

2.3 A ESTRUTURA DE MENSAGENS

A comunicação na rede CAN ocorre através da troca de pacotes de informação denominados *frames*. O protocolo define quatro tipos principais de *frames*: o *Data Frame*, que transporta os dados; o *Remote Frame*, para solicitar a transmissão de dados de outro nó; o *Error Frame*, sinalizado por qualquer nó que detecte uma falha no barramento; e o *Overload Frame*, usado para solicitar um tempo adicional entre mensagens [12].

Para o escopo do projeto CAN-ESP, cujo foco é a transmissão de dados de sensores e comandos, o *Data Frame* é o elemento central. O protocolo CAN especifica dois formatos de *Data Frame*: o Padrão (CAN 2.0A), com um identificador de 11 bits, e o Estendido (CAN 2.0B), com 29 bits. O projeto CAN-ESP adota exclusivamente o formato Estendido para ampliar o espaço de identificação e organização funcional das mensagens, favorecendo a escalabilidade. Essa decisão envolve o *trade-off* de um *overhead* maior por quadro; contudo, tal custo é absorvido pela capacidade de processamento do ESP32, preservando as propriedades temporais do sistema [12].

A estrutura de um *Data Frame* Estendido é composta por múltiplos campos que definem sua prioridade, conteúdo e integridade. A Figura 2.5 detalha a estrutura de ambos os formatos, Padrão (2.0A) e Estendido (2.0B), que se diferenciam principalmente na composição do seu Campo de Arbitragem. Funcionalmente, esses campos podem ser compreendidos em três partes: identificação e priorização; conteúdo e controle; e, validação e conclusão.

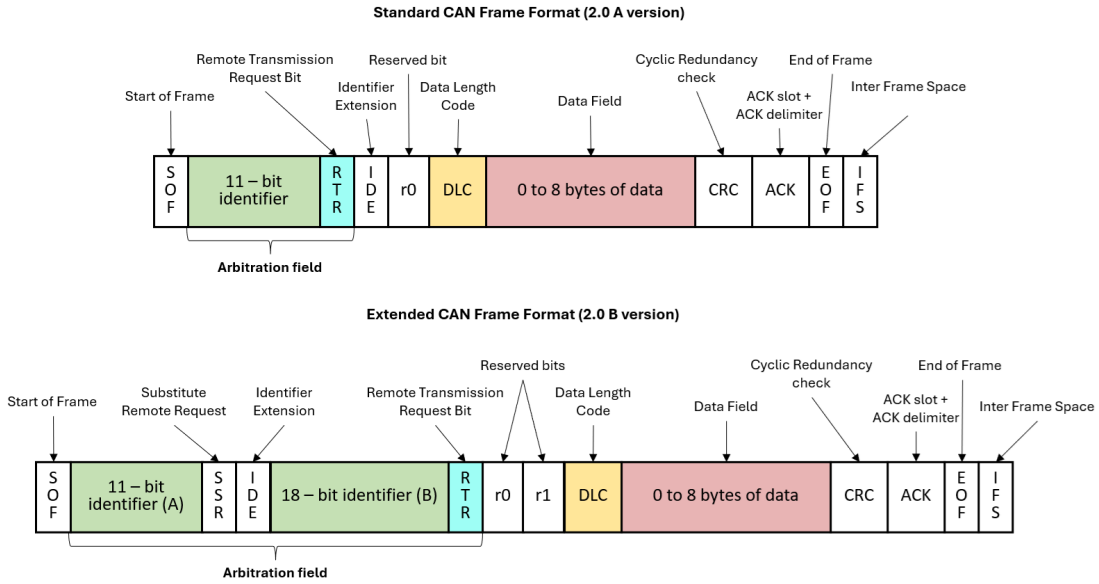

Figura 2.5: Estrutura do *Data Frame* na rede CAN, destacando os campos das versões Padrão e Estendida [3]. Fonte: elaborada pelo autor (2025).

A parte de identificação e priorização é definida pelo *Start of Frame* (SOF), um único bit dominante que sincroniza todos os nós da rede, e pelo *Arbitration Field*. Este campo é fundamental para o gerenciamento do barramento, contendo o Identificador (ID) de 11 ou 29 bits que, como já discutido, define a prioridade da mensagem. Não se trata de um "endereço" de destino, mas de um rótulo que permite ao *hardware* de cada nó arbitrar o acesso ao barramento em tempo real.

A parte de conteúdo e controle é composta pelo *Control Field*, um campo de 6 bits que, entre outras informações, especifica o *Data Length Code* (DLC), indicando quantos bytes de dados a mensagem transporta. Imediatamente após, o *Data Field* contém a carga útil da mensagem, podendo conter de 0 a 8 bytes de dados. É neste campo que as informações dos sensores, atuadores e comandos do sistema CAN-ESP são efetivamente transportadas.

Por fim, a parte de validação e conclusão garante a integridade da transmissão. O *CRC Field* (16 bits) contém uma soma de verificação (*Cyclic Redundancy Check*) que permite aos nós receptores verificarem se a mensagem foi recebida sem corrupção por erros de transmissão. É importante notar, contudo, que este mecanismo não oferece proteção contra modificações maliciosas intencionais, pois não envolve criptografia [26]. Em seguida, o *ACK Field* (2 bits) permite que os nós receptores sinalizem ao transmissor que a mensagem foi recebida corretamente (*Acknowledgment Check*). A transmissão é concluída com o *End of Frame* (EOF), uma sequência de 7 bits recessivos que libera o barramento. Na prática, a combinação destes campos permite um modelo de comunicação robusto e flexível.

Diferente de protocolos com *frames* de tamanho fixo, o comprimento real de uma mensagem CAN no barramento é variável. Essa variação decorre de dois fatores: o tamanho da carga útil (*Data Field*) e o mecanismo de *bit stuffing*. A Tabela 2.2 apresenta a anatomia bit-a-bit do quadro estendido utilizado no CAN-ESP, baseada na norma ISO 11898-1.

Tabela 2.2: Descrição Funcional e Anatômica do *Data Frame* CAN Estendido.

Campo	Tamanho (bits)	Descrição Funcional Estratégica
***Start of Frame* (SOF)**	1	**Sincronização.** Alerta todos os nós sobre o início de uma nova transmissão.
Arbitration Field	12 (Padrão) / 32 (Estendido)	**Priorização.** Contém o Identificador (ID) que define a prioridade da mensagem durante a arbitragem.
Control Field	6	**Controle.** Especifica o tamanho da carga útil (DLC) e reserva bits para futuras expansões.
Data Field	0 a 64	**Carga Útil.** Transporta os dados efetivos da aplicação (leituras de sensores, comandos, etc.).
CRC *Field*	16 (15 + 1 delimitador)	**Detecção de Erros.** Garante a integridade da mensagem contra falhas de transmissão.
ACK *Field*	2 (1 slot + 1 delimitador)	**Confirmação.** Permite que os receptores sinalizem o recebimento válido da mensagem.
***End of Frame* (EOF)**	7	**Finalização.** Sequência de bits recessivos que sinaliza o fim da transmissão e libera o barramento.
IFS (Intermission)	3	**Separação.** Espaço mínimo obrigatório entre quadros.

Um aspecto crítico para o determinismo da rede é o fenômeno do *bit stuffing*. Para garantir a sincronização do relógio dos nós receptores, o controlador insere um bit de polaridade oposta após cada sequência de cinco bits consecutivos de mesmo nível lógico. Este mecanismo é aplicado desde o SOF até o final da sequência de CRC. Consequentemente, o tempo total de ocupação do barramento varia entre o cenário nominal (sem bits de preenchimento) e o cenário de pior caso (*worst-case*), conforme detalhado na Tabela 2.3.

Tabela 2.3: Variação do tamanho do quadro estendido no sistema CAN-ESP.

Dados (Bytes)	Tamanho do Quadro (bits)	Total com IFS (bits)	Pior Caso / Stuffing (bits)
0	64	67	80
8	128	131	160

Essa variação é a explicação teórica para o *jitter* observado nos experimentos práticos deste trabalho. Enquanto o tamanho nominal de um quadro com 8 bytes de dados é de 128 bits, a necessidade do espaço interquadro (*Intermission*) eleva a ocupação mínima para 131 bits. Adicionalmente, a inserção de bits de *stuffing* pode elevar essa ocupação para até 160 bits (incluindo o IFS) em cenários de pior caso, impactando diretamente o cálculo do tempo de resposta (*Worst-Case Response Time* - WCRT).

O sistema CAN-ESP gerencia essa estrutura através da pilha TWAI (*Two-Wire Automotive Interface*), nativa do ESP32 e compatível com a ISO 11898 [25]. A filtragem dessas mensagens, essencial para a eficiência do nó, ocorre via *hardware* através de máscaras de aceitação, permitindo que o processador principal do ESP32 seja notificado apenas por mensagens relevantes, conforme ilustrado na Figura 2.6.

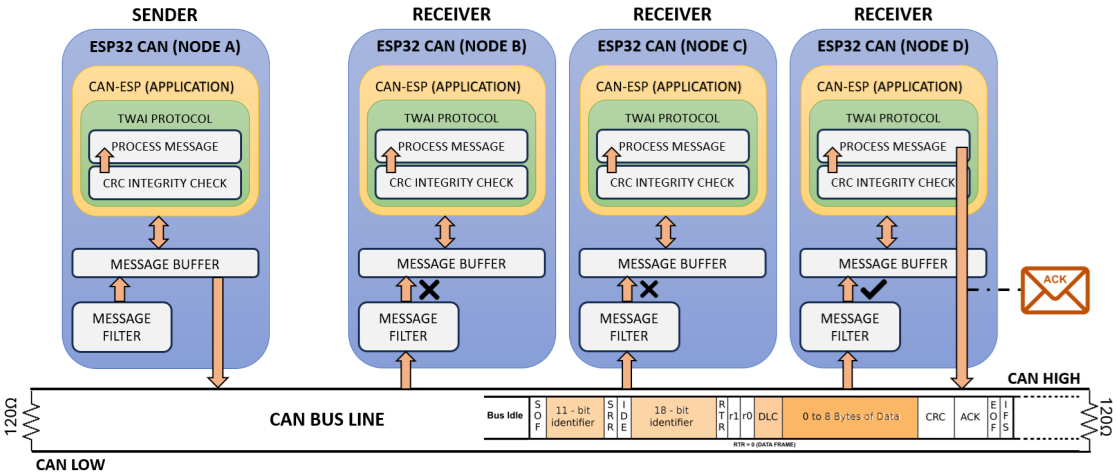

Figura 2.6: Ilustração do processo de transmissão e recepção com filtragem de mensagens no barramento CAN. Fonte: elaborada pelo autor (2025).

A compreensão detalhada desta estrutura de pacotes é, portanto, o pilar teórico fundamental necessário antes de mergulharmos na arquitetura específica do CAN-ESP.

2.4 ATUALIZAÇÃO REMOTA DE *FIRMWARE* OTA E REDES *MESH*

A crescente complexidade dos sistemas embarcados em veículos modernos impõe a necessidade de mecanismos robustos e seguros para a atualização remota de *firmware*. O processo de atualização OTA é fundamental para a correção de falhas, implementação de novas funcionalidades e mitigação de vulnerabilidades de segurança ao longo do ciclo de vida do veículo, eliminando a necessidade de intervenções físicas onerosas e, por vezes, muito difíceis de realizar. A criticidade deste mecanismo é tal que sua segurança e confiabilidade são consideradas pilares para a proteção de sistemas IoT [27]. Contudo, a implementação de um sistema OTA em um ambiente veicular, composto por múltiplas ECUs interdependentes, apresenta desafios únicos de conectividade, confiabilidade e segurança. Soluções monolíticas com frequência não conseguem resolver a heterogeneidade da comunicação, que engloba tanto a interação com a nuvem quanto a comunicação intraveicular.

A funcionalidade de atualização remota do CAN-ESP se baseia em uma arquitetura de rede sem fio. Para tal, foi adotado o protocolo de rede em malha (*mesh*), que possui características de robustez e cobertura superiores às redes Wi-Fi convencionais.

O protocolo ESP-WIFI-MESH, em particular, foi projetado para ser auto-organizável e auto-reparável (*self-organizing and self-healing*), o que significa que a rede pode ser construída e mantida de forma autônoma [28]. Esta capacidade é fundamental para sistemas embarcados que exigem alta disponibilidade e mínima intervenção manual. As redes *mesh* baseadas em infraestrutura Wi-Fi constituem um protocolo de

comunicação que organiza múltiplos nós individuais em uma única rede de área local sem fio (WLAN). Esta arquitetura supera as limitações do Wi-Fi convencional em topologia estrela, onde as estações mantêm exclusivamente uma conexão *upstream* com um ponto de acesso (AP), que por sua vez estabelece múltiplas conexões *downstream* [28]. A Tabela 2.4 mostra as principais diferenças entre a arquitetura Wi-Fi convencional e a Rede *Mesh* Wi-Fi.

Tabela 2.4: Principais diferenciais estruturais entre redes Wi-Fi tradicionais e redes *mesh* Wi-Fi

Parâmetro	Wi-Fi Tradicional	Rede *Mesh* Wi-Fi
Conexões por nó	1 conexão *upstream* (STA)	1 conexão *upstream* (STA) + n conexões *downstream* (softAP)
Topologia	Estrela (centralizada)	Árvore/Grafo acíclico (distribuída)
Encaminhamento	Mono-salto ($h_{\text{max}} = 1$)	Multi-salto ($h_{\text{max}} \geq 2$)
Tolerância a falhas	Baixa (ponto único de falha)	Alta (auto-reparo por reassociação)

Na topologia *mesh*, cada nó implementa uma dualidade funcional, operando simultaneamente como estação (STA) e ponto de acesso (softAP). Especificamente, o nó mantém uma única conexão upstream (em direção à raiz da rede) através de sua interface STA, enquanto pode estabelecer múltiplas conexões *downstream* (em direção às folhas da rede) através de sua interface softAP. Esta operação gera uma topologia arbórea multicamadas, organizada hierarquicamente. Dentro desta hierarquia, os nós são classificados da seguinte forma:

- **Nó Raiz** (*Root Node*): é o nó de topo da hierarquia, servindo como a única interface entre a rede *mesh* interna e uma rede IP externa. Ele se conecta a um roteador Wi-Fi convencional e é responsável por encaminhar o tráfego de e para a rede externa. Só pode haver um nó raiz em uma rede ESP-WIFI-MESH [28].
- **Nós Intermediários** (Intermediate Parent Nodes): são nós que não são a raiz, mas que possuem outros nós (filhos) conectados a eles, formando as camadas intermediárias da árvore.
- **Nós Folha** (Leaf Nodes): são os nós nas extremidades da rede, que não possuem nenhum nó filho.

O protocolo oferece dois modos distintos para definição do nó raiz. No modo de seleção manual, um nó específico é explicitamente configurado como raiz. Este nó estabelece conexão direta com o roteador Wi-Fi externo via interface STA enquanto ativa sua interface softAP para aceitar conexões *downstream* de nós filhos. Os demais nós são configurados como `MESH_NODE`, sem credenciais do roteador, restringindo sua conectividade à rede *mesh* interna.

Alternativamente, o modo de eleição automática implementa um algoritmo iterativo: cada nó candidato escaneia *beacons* Wi-Fi, mede o RSSI com o roteador externo, transmite *beacons* com seu MAC e RSSI, recebe votos dos vizinhos e torna-se raiz se obtiver $\geq 90\%$ dos votos (*threshold* configurável). Falhas no *root* atual acionam *root switching* dinâmico. Ambos culminam na formação de topologia arbórea hierárquica, com o nó raiz mantendo conexão *upstream* externa e *downstream* com filhos, que podem atuar como *softAPs* intermediários [28].

Igualmente importante é a capacidade de auto-reparação (*self-healing*). Se um nó intermediário ou mesmo o nó raiz falhar, os nós que estavam conectados a ele detectarão a perda de conexão e iniciarão um processo para encontrar um novo "pai" na rede, ou, se necessário, eleger um novo nó raiz, restaurando a conectividade da rede de forma autônoma [28]. Esta resiliência é o que torna a tecnologia *mesh* ideal para aplicações críticas como a atualização de *firmware* em um ambiente veicular.

Outra característica fundamental desta topologia é a capacidade de comunicação **multissalto** (*multi-hop*), que permite que pacotes transitem entre nós que estão fora do alcance de rádio direto, utilizando nós intermediários como repetidores. A topologia resultante apresenta, portanto, uma tolerância a falhas superior à arquitetura em estrela principalmente pela capacidade de **auto-reparo** e **reconfiguração dinâmica** (por exemplo, reassociação a um novo nó pai e reorganização hierárquica) quando um nó intermediário falha, restaurando a conectividade ao longo do tempo — ainda que, em uma topologia arbórea, não se explorem múltiplos caminhos simultâneos como em malhas totalmente conectadas [28].

3 TRABALHOS CORRELATOS

A implementação de redes CAN tem sido amplamente investigada na literatura, especialmente no contexto de aplicações automotivas e sistemas embarcados, com diversos estudos explorando soluções para aprimorar a comunicação entre módulos eletrônicos [6]. No entanto, uma análise crítica do recorte revisado nesta dissertação indica um estado da arte heterogêneo, no qual diferentes soluções assumem compromissos relevantes entre custo, desempenho, funcionalidades modernas e rigor na validação experimental. Este capítulo apresenta uma análise temática dos trabalhos correlatos para identificar, de forma sistemática, as lacunas na literatura que motivam e justificam a proposta do CAN-ESP. A Tabela 3.1 oferece um panorama comparativo das principais características das obras aqui discutidas.

Tabela 3.1: Análise Comparativa das Características dos Trabalhos Relacionados.

Trabalhos / Características	**ISO 11898-1**	**Redes Distribuídas**	**Aplicada em EV/ Outros veículos**	**Atualização OTA**	**Wi-Fi & Bluetooth**	**Open Source**	**Baixo Custo**	**Métricas Quantitativas**
Abordagens de Baixo Custo com Arquitetura Fragmentada								
Li et al. (2010) [29]	+	+	+	–	–	–	+	–
Desai et al. (2013) [13]	+	+	+	–	–	–	+	–
Vijaya et al. (2015) [30]	+	+	+	–	–	–	+	–
Wagh et al. (2017) [31]	+	+	+	–	–	–	+	–
Sase et al. (2022) [14]	+	+	+	–	–	–	+	–
Alzahrani et al. (2023) [32]	–	+	+	–	–	–	+	–
Arquiteturas de Controle Complexas e/ou Desconectadas								
Ismail et al. (2015) [33]	+	+	+	–	–	–	+	–
Meah et al. (2020) [16]	+	+	+	–	–	–	–	–
Wang (2021) [34]	–	+	+	–	–	–	–	–
Jiang (2022) [35]	+	+	+	–	–	–	–	–
Salunkhe et al. (2016) [36]	+	+	+	–	+	–	–	–
Kumar et al. (2023) [37]	+	(+)[c]	+	–	(+)	–	–	–
Mohammed et al. (2023) [38]	+	(+)	+	–	–	–	–	–
Soluções Focadas em Conectividade e Estado da Arte								
Chikhale (2018) [26]	+	+	+	–	+	–	–	–
Pham et al. (2022) [39]	+	+	+	–	+	–	+	–
Nasr et al. (2022) [40]	N/A	+	+	+	+	–	+	–
Giron et al. (2023) [41]	+	+	+	–	(+)[d]	–	+	–
Coelho e Silva-Filho (2024) [15]	+	+	+	–	(+)	–	–	+
Shamshiri et al. (2024) [42]	+	+	+	–	(+)	–	+	+
Ammar et al. (2024) [43]	+	+	+	–	–	–	+	+
CAN-ESP (este trabalho)[a]	**+**	**+**	**+**	**+**	**+**	**+**	**++**[b]	**+**

[a] Nota: '+' possui; '–' não possui; '++' custo significativamente menor; '(+)' funcionalidade limitada, não utilizada operacionalmente ou planejada como trabalho futuro; 'N/A' não se aplica.

[b] O custo é considerado significativamente menor ('++') pois a arquitetura se baseia em um SoC ESP32 (ordem de grandeza de USD 5–7 em varejo), enquanto outras abordagens flexíveis frequentemente utilizam plataformas como Raspberry Pi (USD 35+ em varejo) ou *hardware* automotivo/industrial dedicado de custo substancialmente superior. Ressalta-se que valores absolutos variam por região, escala de compra e cadeia de suprimentos; nesta dissertação, a comparação é utilizada como referência relativa de ordem de grandeza.

[c] Arquitetura de 2 nós, sem ECUs múltiplas.

[d] *Hardware* suporta, mas não utilizado operacionalmente.

3.1 O PARADIGMA DO BAIXO CUSTO E SUAS LIMITAÇÕES

A busca por soluções de baixo custo para redes veiculares, embora relevante em sua intenção de democratização tecnológica, apresenta limitações recorrentes quando analisada sob requisitos contemporâneos de desempenho, escalabilidade e manutenção. Trabalhos representativos como o de Sase et al. (2022) [14], que propõem sistemas modulares baseados em Arduino Uno e módulos CAN externos (MCP2515), exemplificam como a fragmentação arquitetural pode impor restrições práticas de desempenho e de funcionalidades modernas. A comunicação via interface serial (SPI) entre o microcontrolador e controladores CAN externos (como o MCP2515) introduz uma latência adicional de processamento (associada à pilha de *software* e ao acesso periférico) que tende a degradar a previsibilidade temporal do sistema. Em particular, análises de *Worst-Case Response Time* (WCRT) em cenários com maior carga de barramento evidenciam que latências adicionais e variações associadas à arquitetura e ao *software* podem impactar o comportamento de tempo real [43]. Em contrapartida, a arquitetura CAN-ESP utiliza o controlador TWAI integrado nativamente ao SoC ESP32, reduzindo gargalos de comunicação externa e permitindo que a biblioteca `can_esp_lib` realize instrumentação com resolução de microssegundos para caracterização de latência e *jitter* no plano de execução.

Problemas de escalabilidade emergem como segunda limitação fundamental: cada novo nó tende a exigir componentes redundantes (MCU, controlador CAN, transceptor e, em alguns casos, interfaces adicionais), o que aumenta o custo por nó e torna a integração física (cabeamento, conectores e montagem) progressivamente mais complexa. Salunkhe et al. (2016) [36] reportam custos da ordem de US$40 por nó em soluções baseadas em Raspberry Pi, enquanto Kumar et al. (2023) [37] discutem como a fragmentação arquitetural pode dificultar a expansão funcional, especialmente quando se busca integrar múltiplos barramentos e microcontroladores heterogêneos.

Três limitações recorrentes permeiam essas soluções: validação predominantemente funcional, frequentemente sem a apresentação de métricas quantitativas de desempenho (como latência, *jitter* ou carga do barramento); desconexão digital sistêmica, em que os veículos operam como "ilhas tecnológicas", sem conectividade sem fio nativa ou capacidade de atualização remota (OTA); e, estagnação tecnológica, com implementações que preservam paradigmas tradicionais e exploram de forma limitada a convergência entre CAN e IoT discutida por Chikhale (2018) [26].

Curiosamente, essas limitações não se restringem à esfera econômica, sendo também observadas em propostas com *hardware* de maior desempenho, como a SAVIPS de Coelho e Silva-Filho (2024) [15]. Ainda que utilize plataformas de alta capacidade computacional (como NVIDIA Jetson Orin e *gateways* dedicados), o recorte analisado nesta dissertação indica que funcionalidades de gerenciamento remoto e atualização de *firmware* em ECUs distribuídas não são tratadas como capacidade nativa da arquitetura.

A evolução deste paradigma pode ser interpretada como uma maior integração do subsistema CAN em um SoC com conectividade nativa, conforme implementado no CAN-ESP. Essa transição reduz a fragmentação de componentes ao empregar um controlador CAN integrado ao SoC; reduz o custo por nó para a ordem de grandeza de US$7; habilita funcionalidades modernas como atualização remota e diagnóstico; e tende a favorecer previsibilidade temporal, em linha com as métricas experimentais de WCRT apresentadas no Capítulo 5.

Mesmo quando plataformas com ESP32 são utilizadas [39], a ausência de uma filosofia arquitetônica centrada em conectividade impede o aproveitamento completo dos recursos nativos disponíveis no *hardware*. Nestes casos, a conectividade Wi-Fi é subutilizada, restringindo-se à transmissão unidirecional de dados de telemetria. Em contraste, a arquitetura CAN-ESP avança o estado da arte ao transformar os nós da rede em agentes ativos capazes de autogestão via rede *mesh*. A implementação de um protocolo de camada de aplicação permite não apenas a leitura de dados, mas a reconfiguração dinâmica e a atualização segura do *firmware*, endereçando a lacuna de 'gerenciabilidade' identificada na literatura. A arquitetura CAN-ESP coloca a conectividade, a modularidade e a manutenção remota como fundamentos centrais do projeto, e não como extensões periféricas.

Portanto, o paradigma tradicional de baixo custo, embora historicamente relevante e ainda útil em contextos educacionais e experimentais, apresenta limitações importantes para atender a demandas contemporâneas associadas a veículos definidos por *software*. O CAN-ESP é proposto como uma alternativa arquitetural integrada, visando conciliar desempenho, conectividade e custo reduzido, e oferecendo um caminho de evolução técnica para além de soluções fragmentadas.

3.2 ARQUITETURAS DE CONTROLE: COMPLEXIDADE, CUSTO E CONECTIVIDADE

Como alternativa às limitações das plataformas de baixo custo, uma vertente distinta da literatura explora o uso de *hardware* mais poderoso. Esta busca por desempenho, no entanto, pode resultar em soluções de maior complexidade e custo, sem necessariamente atender a aspectos críticos de conectividade moderna. Arquiteturas baseadas em computadores industriais ou chips DSP (*Digital Signal Processor*) especializados privilegiam a robustez elétrica, mas, como discutido por Wang (2021) [34] e Jiang (2022) [35], sua natureza com múltiplos subsistemas (chips auxiliares, isoladores, conversores e interfaces adicionais) tende a aumentar a quantidade de interfaces críticas e pontos potenciais de falha[1] e pode contribuir para a formação de "ilhas tecnológicas" com conectividade limitada.

Mesmo soluções que utilizam microcontroladores de alto desempenho, como em Meah et al. (2020) [16], ou plataformas de ponta com processadores como o NVIDIA Jetson Orin, como na SAVIPS de Coelho e Silva-Filho (2024) [15], persistem com problemas de isolamento funcional. A ausência de mecanismos de atualização OTA, uma lacuna criticada por Nasr et al. (2022) [40], pode comprometer não apenas a manutenção contínua, mas também a segurança cibernética, ao impedir a correção rápida de vulnerabilidades, um risco crítico considerando que falhas de *software* podem exigir intervenções urgentes em campo.

Outro padrão recorrente é o acoplamento de *Single Board Computers* (SBCs), como o Raspberry Pi, a microcontroladores mais simples via módulos CAN externos. Esta abordagem, vista em Salunkhe et al. (2016) [36] e Kumar et al. (2023) [37], enfrenta desafios intrínsecos de escalabilidade, como o alto consumo energético (5–10W por nó) e a dificuldade de prover garantias de tempo real típicas de sistemas *hard real-time*. Além disso, a comunicação interna mediada por interfaces periféricas (e.g., SPI) pode

[1] Mesmo em um nó com SoC integrado, permanecem componentes essenciais (e.g., transceptor e regulação de energia). A comparação aqui destaca que arquiteturas com controlador externo e múltiplas pontes de comunicação adicionam interfaces que elevam a complexidade de integração e diagnóstico.

introduzir latências adicionais e variabilidade temporal, conforme discutido por Salunkhe et al. (2016) [36]. Nessas configurações, a rede CAN pode ser empregada apenas como canal de troca de mensagens sem explorar plenamente seu potencial como barramento distribuído em múltiplos nós.

O ápice desta complexidade manifesta-se em arquiteturas híbridas como a de Mohammed et al. (2023) [38], onde a integração de quatro componentes heterogêneos (laptop, dois Arduinos, rádio dedicado) com três protocolos distintos (CAN, I²C, NRF24L01) resulta em uma arquitetura com múltiplos pontos de falha interdependentes, onde a falha de qualquer componente compromete o funcionamento global. Esta fragilidade sistêmica é diretamente proporcional ao número de interfaces críticas, conforme evidenciado nos testes de campo do trabalho.

Diante deste cenário, a arquitetura CAN-ESP emerge como contraponto estrutural. Ao integrar controlador CAN nativo, conectividade Wi-Fi e mecanismos para OTA em um único SoC, a solução reduz pontos de falha e custos em mais de 80%[2] em relação a soluções baseadas em SBCs. Como demonstrado por Ammar et al. (2024) [43], arquiteturas integradas permitem manter determinismo temporal com latências mais baixas e previsíveis. Mesmo trabalhos que utilizam o ESP32, como o de Pham et al. (2022) [39], não exploram seu potencial bidirecional para gerenciamento, evidenciando que a arquitetura CAN-ESP propõe uma reestruturação conceitual que posiciona a conectividade como elemento central, ocupando um espaço estratégico único no estado da arte, conforme sintetizado visualmente na Figura 3.1. O diagrama evidencia a lacuna de soluções de baixo custo e alta conectividade, espaço que o CAN-ESP se propõe a preencher.

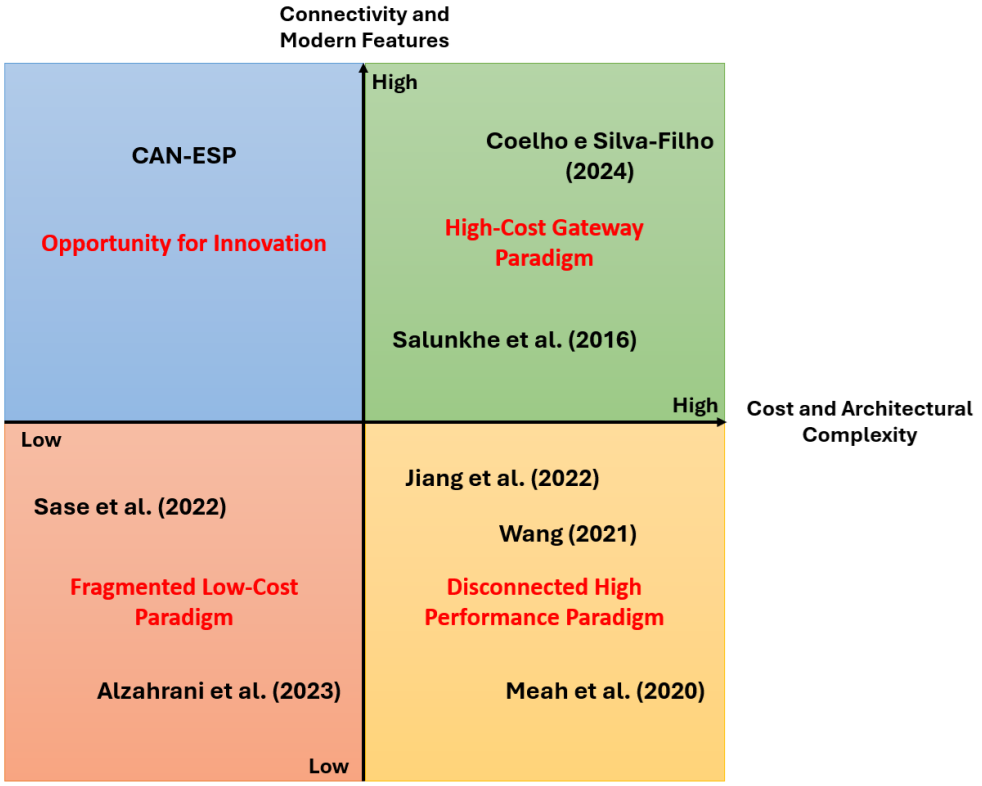

Figura 3.1: Quadrante analítico do estado da arte, posicionando as arquiteturas de redes veiculares segundo seu custo/complexidade e nível de conectividade/funcionalidades modernas. Fonte: elaborada pelo autor (2025).

[2]Comparação baseada no custo de um nó CAN-ESP (aprox. USD 7) versus um nó baseado em Raspberry Pi + Módulo CAN (aprox. USD 40) conforme análise de Salunkhe et al. (2016).

3.3 A FRONTEIRA DA CONECTIVIDADE E A LACUNA DE VALIDAÇÃO

A integração de tecnologias IoT com barramentos CAN, reconhecida como fronteira crítica para inovação em veículos elétricos [26], enfrenta desafios de implementação que transcendem a mera conectividade superficial. Trabalhos como Pham et al. (2022) [39] e Salunkhe et al. (2016) [36] evidenciam uma limitação recorrente: mesmo utilizando SoCs modernos (ESP32) ou SBCs (Raspberry Pi), suas arquiteturas restringem-se à telemetria predominantemente unidirecional, sem mecanismos completos para gestão remota ou atualização OTA. Esta característica tem implicações operacionais, pois a ausência de canais bidirecionais para atualização remota pode exigir intervenções físicas para correção de falhas de segurança ou defeitos críticos, aumentando indisponibilidade e custo operacional, conforme discutido por Nasr et al. (2022) [40]. Paradoxalmente, mesmo soluções dedicadas a OTA como Nasr et al. (2022) [40] frequentemente dependem de *gateways* externos, mantendo uma separação entre conectividade e a arquitetura distribuída intraveicular.

A lacuna mais crítica, contudo, reside na validação empírica insuficiente. Grande parte das implementações analisadas - desde sistemas baseados em Arduino [14] até arquiteturas industriais robustas [34] - limita-se à verificação funcional em cenários controlados, não abordando as métricas quantitativas essenciais. Esta carência torna-se particularmente perigosa em aplicações tempo-críticas, pois a inexistência de análises de *Worst-Case Response Time* (WCRT) ou medições de *jitter* inviabiliza a certificação de sistemas onde decisões milissegundais regulam frenagem assistida ou controle de torque, expondo usuários a riscos sistêmicos. Como evidenciam Ammar et al. (2024) [43], a análise formal de tempo de resposta não é luxo acadêmico, mas requisito fundamental para sistemas embarcados de missão crítica.

Neste contexto, o CAN-ESP distingue-se por abordar simultaneamente ambas as lacunas. Sua arquitetura integra OTA criptografado como capacidade nativa dos nós da rede, eliminando a necessidade de intervenções físicas para atualizações críticas. Mais significativamente, ele supera a tradição de validações superficiais. Conforme será detalhado no Capítulo 5, a arquitetura foi submetida a uma bateria de testes, incluindo a análise de desempenho sob múltiplos cenários de carga (20%, 50%, 80% e 95%), a medição contínua de *jitter* em mensagens periódicas e a determinação empírica do WCRT para comandos críticos. Esta abordagem multivariada, ausente em trabalhos como Shamshiri et al. (2024) [42] que se limitam a cenários únicos, fornece dados replicáveis e robustos que validam não apenas funcionalidade, mas confiabilidade operacional em condições reais. Ao unir conectividade intrínseca com validação experimental rigorosa, o CAN-ESP estabelece um novo paradigma para redes veiculares de baixo custo, demonstrando que agilidade de *software* e confiabilidade temporal não precisam ser mutuamente exclusivas.

3.4 SÍNTESE CRÍTICA E CONTRIBUIÇÕES

A análise aprofundada do estado da arte revela um campo de pesquisa marcado por *trade-offs* significativos. De um lado, o paradigma do baixo custo fragmentado (Seção 3.1) oferece acessibilidade econômica às custas de desempenho limitado, conectividade restrita e maior fragilidade arquitetural. De outro, arquiteturas de alto desempenho com conectividade limitada (Seção 3.2) buscam robustez computacional, mas podem resultar em soluções de alta complexidade, elevado custo e com suporte limitado a gerenciamento

remoto. Transversalmente a essas abordagens, duas lacunas críticas permanecem recorrentes no recorte analisado: a ausência de uma plataforma com mecanismos integrados de atualização remota de *firmware* (OTA) em ECUs distribuídas e a necessidade de maior validação de desempenho baseada em métricas quantitativas e rigorosas (Seção 3.3).

É nesse contexto de desafios multidimensionais que o CAN-ESP se insere como contribuição original deste trabalho. Esta proposta oferece uma arquitetura que busca resolver o paradoxo identificado, sintetizando vantagens que usualmente aparecem separadas na literatura revisada. Dentre os trabalhos analisados nesta dissertação, o CAN-ESP é o que reúne simultaneamente os cinco pilares propostos na introdução: (1) baixo custo, favorecendo replicação em larga escala; (2) conformidade com o padrão ISO 11898, incluindo suporte a identificadores estendidos de 29 bits; (3) capacidade de atualização remota OTA com mecanismos de segurança e confiabilidade descritos no Capítulo 4; (4) arquitetura de *hardware* e *software* de código aberto, com bibliotecas próprias e documentação pública; e (5) uma avaliação de desempenho quantitativa e rigorosa, demonstrada experimentalmente no Capítulo 5.

É imperativo destacar que, embora diversos estudos utilizem plataformas de *hardware* aberto, há escassez de trabalhos que disponibilizam o código-fonte integral de forma organizada, documentada e replicável. O CAN-ESP adota a filosofia open source de maneira integral, oferecendo acesso irrestrito ao código da biblioteca `can_esp_lib` (núcleo lógico da aplicação). Essa transparência reforça a reprodutibilidade científica esperada e alinha a solução aos princípios de ciência aberta e engenharia reprodutível, promovendo auditabilidade técnica e facilitando evolução incremental por terceiros.

Essa convergência de critérios técnicos, conceituais e metodológicos posiciona o CAN-ESP como uma alternativa promissora frente às abordagens tradicionais em redes embarcadas veiculares, visando superar as dicotomias ainda presentes na literatura e oferecendo uma solução escalável, conectada, confiável e economicamente viável.

4 CAN-ESP: UMA ARQUITETURA DE BAIXO CUSTO E CÓDIGO ABERTO COM SUPORTE A OTA PARA VEÍCULOS ELÉTRICOS

Em resposta às lacunas de custo, complexidade, conectividade e validação identificadas no estado da arte, esta dissertação apresenta a CAN-ESP: uma arquitetura de rede veicular distribuída, de baixo custo e alto desempenho. A filosofia do CAN-ESP é fundamentada na utilização de um SoC moderno e de baixo custo para criar uma rede de nós homogêneos, inteligentes e nativamente conectados, ampliando as funcionalidades encontradas na literatura. A validação desta arquitetura é realizada através da implementação de um protótipo funcional em um Veículo Terrestre Não Tripulado (UGV), o projeto CELINA, desenvolvido em paralelo a este trabalho. Este capítulo detalha a visão geral da arquitetura, os componentes de *hardware* e *software*, o modelo de comunicação e as funcionalidades de confiabilidade e atualização OTA que definem a contribuição do CAN-ESP.

4.1 FILOSOFIA E VISÃO GERAL DA ARQUITETURA

A arquitetura do CAN-ESP, ilustrada na Figura 4.1, fundamenta-se no modelo de barramento distribuído multimestre, conforme especificado pela norma ISO 11898 [18]. Cada nó (ECU) é composto por um SoC ESP32 (que integra o controlador TWAI) e um transceptor CAN, comunicando-se através de um barramento linear único. A escolha por esta topologia foi uma decisão de *design* deliberada para abordar diretamente os desafios de custo e complexidade vistos em arquiteturas centralizadas ou heterogêneas. Os princípios que guiaram o desenvolvimento foram: a redução de cabeamento através de um barramento compartilhado; a modularidade e escalabilidade, permitindo a fácil adição de novas ECUs; e a robustez operacional pela ausência de uma ECU central de controle como ponto único de falha lógica, embora o barramento físico (cabeamento, conectores, transceptores e terminações) permaneça como elemento crítico para a disponibilidade da comunicação [6, 18]. A rede opera com o protocolo CAN 2.0B (*extended frame*) a uma velocidade de 1 Mbps, garantindo a largura de banda necessária para aplicações de controle em tempo real.

O componente central que viabiliza a filosofia do CAN-ESP é o SoC ESP32 WROOM-32. Diferentemente de abordagens que utilizam microcontroladores tradicionalmente aplicados em redes veiculares, a escolha do ESP32 permitiu a fusão da robusta comunicação CAN com a conectividade sem fio (Wi-Fi) em um único componente de baixo custo. O sistema utiliza o controlador TWAI (*Two-Wire Automotive Interface*), a implementação de hardware do protocolo CAN 2.0B nativa da arquitetura ESP-IDF, garantindo conformidade na camada de enlace com CAN 2.0B e compatibilidade com a família ISO 11898 quando empregado com transceptores e camada física adequados [7, 18]. Esta integração nativa, conforme ilustrado no comparativo da Figura 4.2, é o que permite ao CAN-ESP oferecer, além da comunicação veicular, funcionalidades avançadas de gerenciamento, como as atualizações de *firmware* OTA, que serão detalhadas

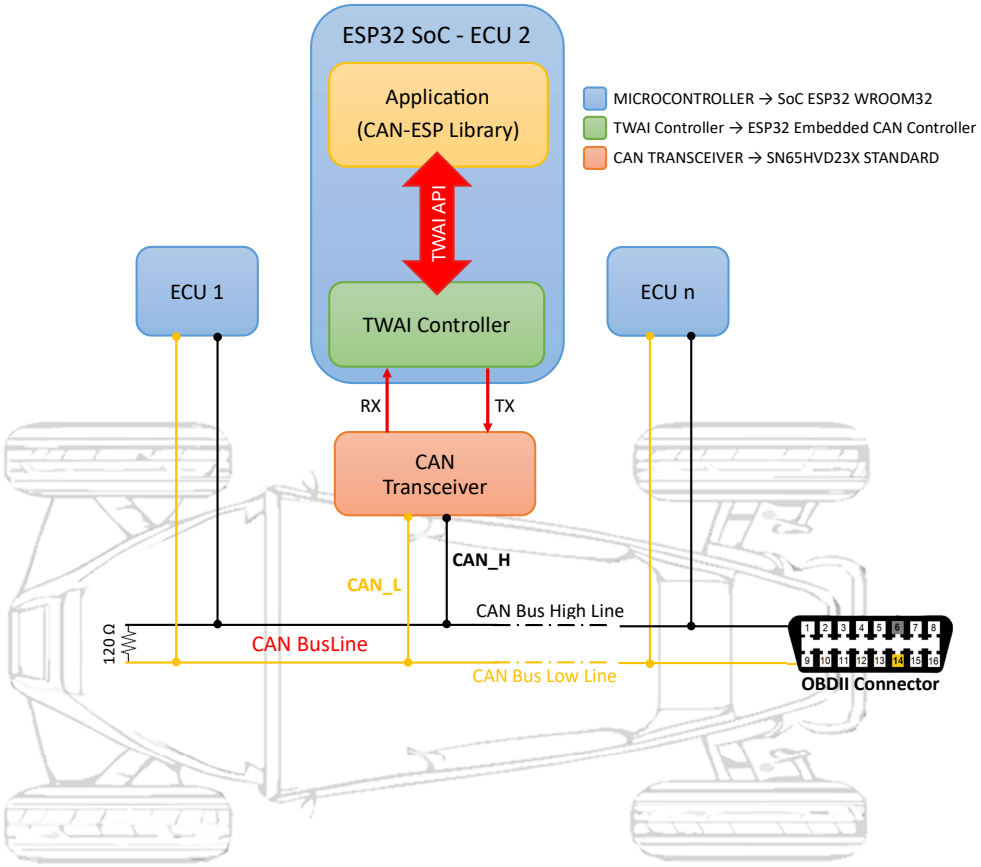

Figura 4.1: Visão geral da arquitetura distribuída multimestre da rede CAN-ESP. Fonte: elaborada pelo autor (2025).

na Seção 4.5. Na figura, é feito um comparativo entre a arquitetura fragmentada (A), comum na literatura, com múltiplos CIs discretos e interfaces de comunicação adicionais (e.g., SPI), e a arquitetura integrada (B) do CAN-ESP, que unifica MCU e controlador CAN em um único SoC, reduzindo custo, complexidade e pontos de falha. O código-fonte completo da biblioteca desenvolvida, `can_esp_lib`, está disponível publicamente para consulta e replicação[1].

4.2 A ARQUITETURA DE *HARDWARE*

A robustez, o baixo custo e a flexibilidade da rede CAN-ESP são um reflexo direto da escolha estratégica de seus componentes de *hardware*. Cada nó da rede (ECU) é construído sobre uma base de *hardware* homogênea, garantindo a reprodutibilidade e a escalabilidade do sistema. Os componentes foram selecionados não apenas por seu custo, mas por sua capacidade de formar uma arquitetura otimizada para integração funcional. A Tabela 4.1 resume os principais componentes utilizados no protótipo.

[1] O repositório do projeto encontra-se em: <https://github.com/danilo-moura-pereira/CAN-ESP>

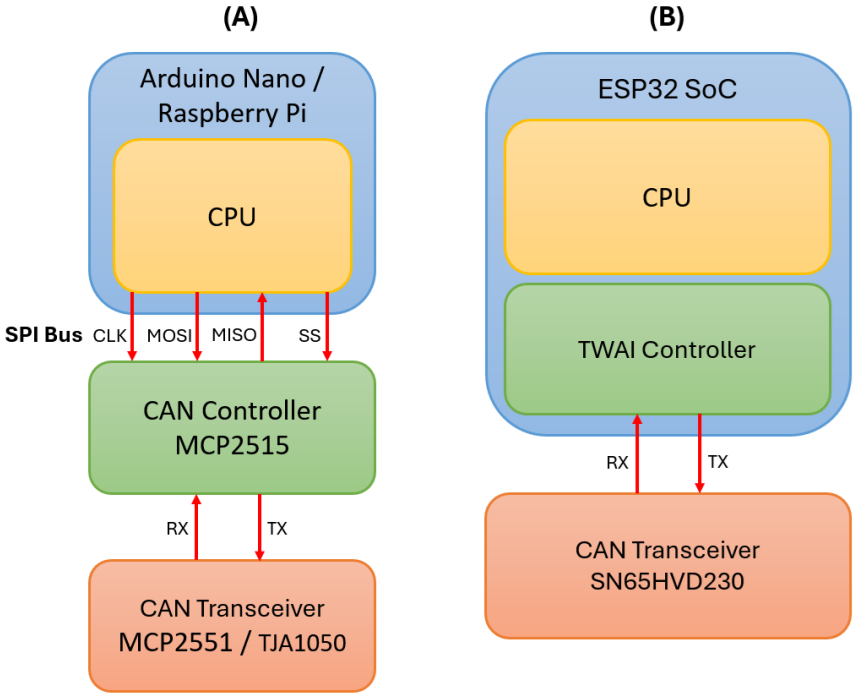

Figura 4.2: Comparativo de paradigmas arquiteturais. Fonte: elaborada pelo autor (2025).

4.2.1 Componentes Principais

O *hardware* de cada nó CAN-ESP é composto por dois elementos fundamentais: o SoC que atua como unidade de processamento e o transceptor que faz a interface com o meio físico.

O cérebro e o coração de cada nó da rede é o SoC ESP32 WROOM-32. A escolha deste microcontrolador foi a decisão de *design* fundamental para o projeto. Equipado com um microprocessador Xtensa® Dual-Core 32-bit LX6, operando a até 240 MHz, com 520 KB de RAM e 4 MB de memória *flash*, ele oferece um poder de processamento adequado para aplicações de controle veicular em tempo real [25]. Sua principal vantagem sobre microcontroladores tradicionais é a sua arquitetura integrada, que embute não apenas o processador, mas também um controlador CAN 2.0B (TWAI) e um rádio de 2.4 GHz com suporte a Wi-Fi. Essa dupla funcionalidade resolve o dilema custo × conectividade identificado no Capítulo 3, eliminando a necessidade de componentes externos para comunicação remota.

Para a interface com o barramento físico, cada nó utiliza o transceptor CAN **SN65HVD230**. Este componente atua como a ponte entre o mundo digital do controlador TWAI e o mundo elétrico do barramento. Ele é responsável por converter os sinais lógicos de transmissão (TX) do controlador em sinais elétricos diferenciais para o barramento, e converter os sinais diferenciais do barramento de volta para um sinal lógico de recepção (RX) para o controlador, garantindo os níveis de tensão e a imunidade a ruído exigidos pelo ambiente automotivo. O barramento em si foi implementado utilizando um cabo de par trançado do tipo UTP CAT-5 (24 AWG), uma solução de baixíssimo custo e alta disponibilidade. Para o comprimento reduzido do barramento no protótipo, a integridade de sinal mostrou-se adequada, corroborada pelos baixos índices de retransmissão e pela ausência de estados críticos do controlador na avaliação experimental (Capítulo 5).

Tabela 4.1: Componentes Principais do Protótipo CAN-ESP.

Categoria	Componente e Descrição
Componentes Centrais	• **ESP32 WROOM-32**: SoC com CPU Xtensa® Dual-Core 32-bit, clock de até 240 MHz, 520 KB RAM, 4 MB flash e controlador CAN 2.0B integrado. • **SN65HVD230**: transceptor CAN para a interface com o barramento físico.
Sistema de Tração	• **Motor Elétrico**: 2x motor *brushless* de 8", 36V, 350W (de Hoverboard). • **Controladora de Motor**: 2x controladora *brushless* de 36V, 18A, 350W.
Interface do Condutor	• **Acelerador**: pedal eletrônico original do Volkswagen UP. • **Sensor de Freio**: sensor de efeito *hall* 49E com ímã de neodímio. • **Motor de Direção**: motor universal de vidro elétrico, 12V. • **Display**: módulo TFT LCD de 1.8"com interface SPI.
Ferramentas de Análise	• **CANable v2.0**: interface USB-para-CAN para monitoramento. • **Analisador Lógico**: SCM 24 MHz, 8 canais, para análise de sinal.

Finalmente, para garantir a integridade do sinal e mitigar reflexões de onda na linha de transmissão, em plena conformidade com a norma ISO 11898-2, o barramento é terminado em ambas as suas extremidades com **resistores de precisão de 120** Ω [18]. A ausência desses componentes poderia levar à degradação do sinal, especialmente em redes de alta velocidade como a de 1 Mbps utilizada neste projeto, resultando em erros de comunicação e comprometendo a confiabilidade de toda a rede.

4.2.2 Componentes Periféricos e Configuração das ECUs do Protótipo

Para validar a arquitetura CAN-ESP em um cenário operacional, foi desenvolvido um protótipo funcional composto por cinco Unidades de Controle Eletrônico (ECUs) homogêneas, que orquestram um conjunto de sensores e atuadores específicos do veículo de testes (UGV CELINA). A Figura 4.3 ilustra as cinco ECUs conectadas ao barramento CAN, onde cada ECU é mostrada com seus respectivos periféricos (sensores e atuadores), e as setas no barramento representam o fluxo das principais mensagens de controle e status da rede. A seguir, a função de cada ECU é detalhada dentro de seu respectivo domínio funcional.

4.2.2.1 Domínio da Interface Homem-Máquina (IHM)

Este domínio compreende as ECUs responsáveis por traduzir as intenções do condutor em dados para a rede e por fornecer *feedback* sobre o estado do veículo.

ECU de Controle da Aceleração. Atua como a principal interface de comando de torque. Sua função é ler o sinal analógico do pedal de acelerador eletrônico (original do Volkswagen UP, 2020) e convertê-lo em mensagens CAN de Torque Requerido (ID `0x002`), que são publicadas no barramento para serem consumidas pelo domínio do trem de força.

ECU de Controle do Freio. Responsável pela detecção da intenção de frenagem, esta ECU monitora um sensor de efeito Hall 49E. Ao detectar a ativação do freio, a ECU transmite imediatamente a mensagem

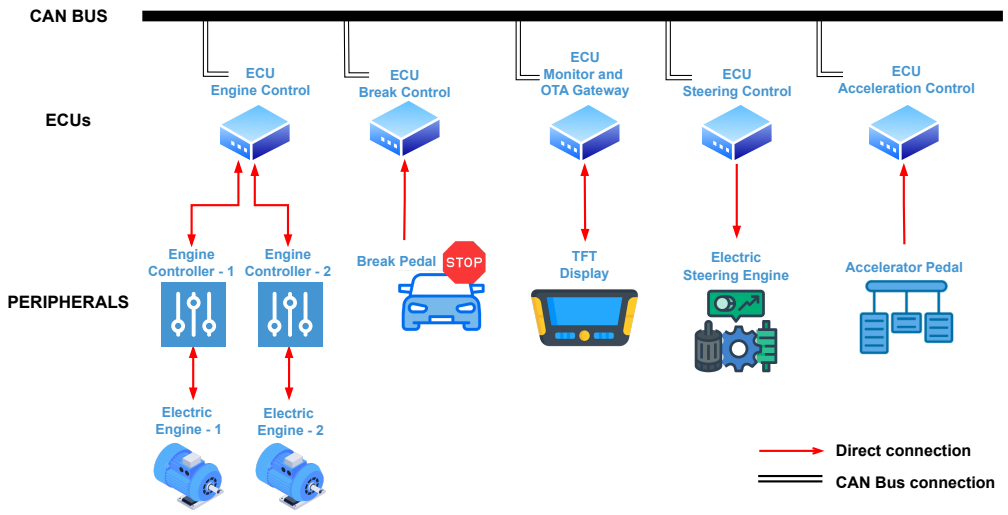

Figura 4.3: Diagrama de Blocos da Arquitetura Funcional do Protótipo. Fonte: elaborada pelo autor (2025).

Pressão de Freio (ID `0x001`) por evento, com a mais alta prioridade na rede, garantindo que esse comando crítico prevaleça na arbitragem do barramento e seja entregue com latência mínima, conforme os princípios de tempo real discutidos na Seção 2.1.3 e a hierarquia de prioridades definida na Tabela 4.2.

ECU de Controle da Direção. Gerencia a atuação da direção assistida. Ela lê a referência de direção (potenciômetro instalado na base do volante) e a converte em sinais PWM para um motor universal de vidro elétrico (12V), adaptado para atuar sobre a coluna de direção do UGV. Adicionalmente, publica o estado de direção na rede por meio da mensagem Ângulo do Volante (ID `0x003`), permitindo monitoramento e integração com demais funções do sistema (Tabela 4.2).

4.2.2.2 Domínio do Trem de Força e Atuação (*Powertrain*)

Este domínio é o coração do sistema de propulsão do veículo, responsável por converter os comandos da rede em movimento.

ECU de Controle do Motor. Esta é a unidade central de atuação. Ela "escuta" o barramento, aguardando mensagens de Torque Requerido (ID `0x002`) e Pressão de Freio (ID `0x001`). Com base nesses dados, ela calcula e envia os sinais de acionamento (PWM) para as duas controladoras de motor *brushless* (36V, 18A, 350W). Estas controladoras, por sua vez, gerenciam a potência entregue aos dois motores elétricos *brushless* de 8 polegadas que tracionam o veículo. Esta ECU também é responsável por ler dados de telemetria dos motores (velocidade de rotação e temperatura), publicando-os no barramento para o domínio de *gateway* e monitoramento.

4.2.2.3 Domínio de *Gateway* e Monitoramento

Esta ECU atua como o nó de convergência da arquitetura, servindo como a ponte entre a rede intraveicular e o operador (ou redes externas).

ECU de Monitoramento e *Gateway* OTA. Possui uma dupla função crítica. Como monitoramento, ela lê mensagens de status do barramento (velocidade, temperatura, etc.) e as exibe para o operador em um display TFT LCD de 1.8 polegadas. Como *gateway*, ela funciona como o *root node* (nó raiz) para as atualizações OTA. Através de seu rádio Wi-Fi, ela se conecta a um servidor MQTT para verificar a disponibilidade de novas versões dos *firmwares*, efetua o download via servidor HTTPS e as distribui para as demais ECUs através de uma rede *mesh*, funcionalidade que será detalhada na Seção 4.5.

4.2.2.4 Plataforma de Testes e Diagnóstico

A validação e a depuração da arquitetura de *hardware* foram suportadas por uma bancada de testes e ferramentas específicas para monitoramento da rede em tempo real.

Uma placa CANable v2.0 serviu como interface USB/CAN para monitoramento e injeção de tráfego utilizando os *softwares* de código aberto CANgaroo e SavvyCAN (analisadores de barramento CAN). Baseada no microcontrolador STM32F072 e controlador CAN MCP2515, esta interface permite captura passiva de frames em tempo real a 1 Mbps com *timestamps* de 1 μs, decodificação de identificadores estendidos (29 bits) e transmissão ativa de tráfego sintético de baixa prioridade para simular a saturação do barramento durante os testes de estresse aplicados no Capítulo 5. Os *softwares* CANgaroo e SavvyCAN, apresentam visualização gráfica do tráfego no barramento, filtros por ID e exportação para análise estatística (dentre outros recursos), sendo essenciais para caracterizar métricas como a carga do barramento, a latência de mensagens críticas (IDs `0x001`, `0x002`) e detecção de anomalias durante os cenários de estresse (S1/S4).

Para uma análise mais profunda da integridade da camada física, um analisador lógico de 8 canais (SCM 24 MHz) foi empregado para inspecionar os sinais elétricos diferenciais CAN_H/CAN_L em baixo nível. Este instrumento capturou simultaneamente os sinais TX/RX do transceptor SN65HVD230, permitindo verificar: (i) níveis de tensão corretos (3,5 V dominante, 2,5 V recessivo); (ii) ausência de *overshoot/undershoot* indicativos de reflexões; (iii) integridade do *bit stuffing*; e (iv) simetria temporal dos bits a 1 Mbps.

4.3 MODELO DE COMUNICAÇÃO E FLUXO DE DADOS

A funcionalidade da arquitetura CAN-ESP não reside apenas em seus componentes de *hardware*, mas fundamentalmente no modelo de comunicação que orquestra a interação entre os nós da rede. Esta seção detalha o protocolo de aplicação desenvolvido para o CAN-ESP, incluindo o mapeamento de mensagens, a lógica de prioridades e os principais ciclos de controle.

4.3.1 Mapeamento de Mensagens e Prioridades

O modelo de comunicação do CAN-ESP é definido por um protocolo de aplicação customizado, construído sobre a camada de enlace de dados especificada pela norma ISO 11898 [18]. Este protocolo explora a estrutura de *frames* estendidos de 29 bits — correspondente ao padrão CAN 2.0B [7] — com o objetivo de criar um sistema de mensagens robusto, onde a prioridade de cada sinal é determinada diretamente pelo valor numérico de seu identificador (ID). Esta abordagem garante que o acesso ao barramento seja determinístico e que as funções críticas do veículo sempre tenham precedência, conforme os princípios adotados em redes de tempo real [18].

A lógica de atribuição dos IDs foi estruturada em faixas funcionais, utilizando valores hexadecimais. Mensagens com prefixos menores (e.g., `0x001`) possuem maior prioridade e são reservadas para os comandos mais críticos de segurança e controle do veículo. Ressalta-se que, embora as mensagens sejam transmitidas no formato estendido (29 bits), os identificadores são apresentados nesta dissertação em hexadecimal sem zeros à esquerda; assim, por exemplo, `0x001` corresponde ao identificador estendido `0x00000001` [7, 18]. Prefixos maiores são delegados a funções de menor prioridade, como telemetria e diagnóstico.

A especificação completa deste protocolo está detalhada na Matriz de Comunicação da Tabela 4.2. Esta tabela é o documento técnico central do modelo de comunicação, definindo o "dicionário" de sinais da rede CAN-ESP.

Tabela 4.2: Matriz de Comunicação do Protocolo CAN-ESP.

Função / Sinal	ID (Hex)	Prioridade	Tipo / Período	DLC (Bytes)	Origem (ECU)	Consumidores (ECU)
Pressão de Freio	`0x001`	**1 (Máxima)**	Periódica / 10 ms	1	Controle do Freio	Controle do Motor
Torque Requerido	`0x002`	2 (Crítica)	Periódica / 50 ms	2	Controle da Aceleração	Controle do Motor
Ângulo do Volante	`0x003`	2 (Crítica)	Periódica / 20 ms	2	Controle da Direção	Controle da Direção, Monitoramento
Velocidade do Motor	`0x010`	3 (Alta)	Periódica / 100 ms	4	Controle do Motor	Monitoramento, Controle do Freio
Temperatura do Motor	`0x011`	3 (Alta)	Periódica / 200 ms	2	Controle do Motor	Monitoramento
Posição do Acelerador	`0x101`	4 (Normal)	Periódica / 50 ms	2	Controle da Aceleração	Monitoramento
Demanda de Potência	`0x102`	4 (Normal)	Periódica / 50 ms	2	Controle da Aceleração	Monitoramento
Status do Motor	`0x500`	5 (Baixa)	Periódica / 1 s	1	Controle do Motor	Monitoramento
Velocidade do Veículo	`0x501`	5 (Baixa)	Periódica / 100 ms	2	Controle do Motor	Monitoramento
Erros de Sistema (DTC)	`0x601`	6 (Diagnóstico)	Sob Demanda	8	Variável	Ferramenta Externa
Leitura de Parâmetros	`0x602`	6 (Diagnóstico)	Sob Demanda	8	Ferramenta Externa	Variável
Status de Comunicação	`0x603`	6 (Diagnóstico)	Periódica / 1 s	8	Monitoramento	Todas

A análise da matriz revela decisões cruciais para a segurança e eficiência do sistema. A mensagem de Pressão de Freio (ID `0x001`) possui a prioridade máxima na rede, sendo transmitida por evento para garantir uma resposta imediata prevalecendo sobre qualquer outra comunicação. Logo em seguida, os comandos de Torque Requerido (ID `0x002`) e Ângulo do Volante (ID `0x003`) formam o segundo nível de criticidade, operando de forma periódica para garantir um controle suave e contínuo. As demais mensagens seguem essa mesma lógica. Esta combinação de sinais por evento para ações de segurança e sinais periódicos para controle em malha fechada é uma prática consolidada na engenharia de sistemas automotivos para otimizar a largura de banda e garantir a previsibilidade da rede [6, 23].

4.3.2 Ciclos de Controle e Fluxo de Dados Operacional

O comportamento dinâmico do veículo ocorre a partir da troca de mensagens entre as ECUs, formando múltiplos laços de controle. A Figura 4.4 ilustra as principais interações que controlam a operação do sistema, desde os comandos iniciados por comportamentos do condutor até a atuação nos motores e o *feedback* para o painel de instrumentos do veículo.

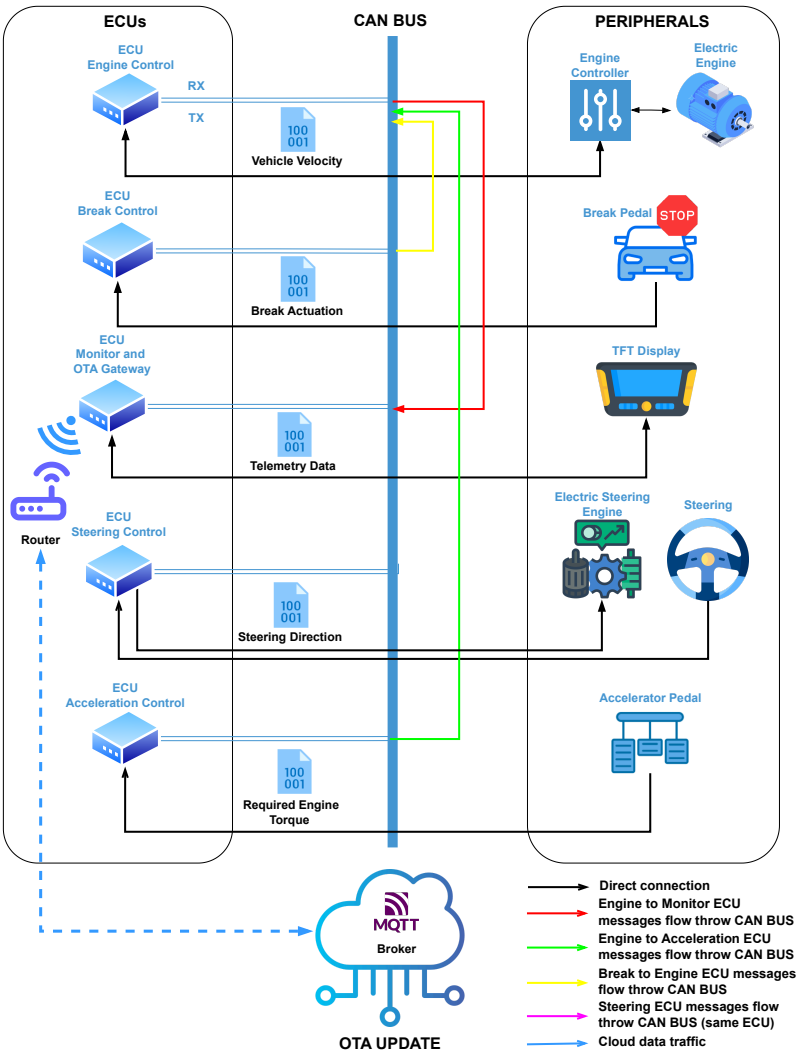

Figura 4.4: Diagrama de Fluxo de Informações da Rede CAN-ESP. Fonte: elaborada pelo autor (2025).

O laço de controle de tração é iniciado pela ECU de Controle da Aceleração. A cada 50 ms, esta ECU lê a posição do pedal e transmite a mensagem Torque Requerido (ID `0x002`). Sendo uma mensagem de prioridade crítica (nível 2), ela é despachada de forma previsível no barramento. A ECU de Controle do Motor, principal consumidora desta mensagem, captura a informação e calcula o ciclo de trabalho do sinal PWM enviado às controladoras dos motores. Para fechar a malha de controle, a própria ECU de Controle do Motor mede a rotação das rodas e publica no barramento a Velocidade do Motor (ID `0x010`) e a Velocidade do Veículo (ID `0x501`), que são consumidas pela ECU de Monitoramento e *Gateway* OTA para exibição no painel.

O laço de controle de frenagem, por sua vez, é acionado por evento. Ao detectar o acionamento do pedal de freio, a ECU de Controle do Freio transmite imediatamente a mensagem Pressão de Freio (ID `0x001`). Por possuir o ID de menor valor numérico, esta mensagem tem a prioridade absoluta na rede (nível 1). Isso significa que, caso a ECU de Controle da Aceleração tente transmitir sua mensagem de torque (ID `0x002`) no mesmo instante, a mensagem de freio (ID `0x001`) vencerá a arbitragem do barramento sem atraso adicional de arbitragem, garantindo que a ordem de parada se sobreponha à de aceleração. Ressalta-se, contudo, que o CAN é não preemptivo: uma mensagem de alta prioridade não interrompe um quadro já em transmissão, estando sujeita, no pior caso, ao bloqueio por um quadro de menor prioridade iniciado imediatamente antes [7, 18].

O laço de controle da direção opera a partir da referência medida localmente na ECU de Controle da Direção. Essa ECU lê os valores do potenciômetro instalado na base do volante, atua sobre o motor elétrico adaptado à barra de direção do veículo por meio de sinais PWM e, em paralelo, publica a mensagem Ângulo do Volante (ID `0x003`) a cada 20 ms para fins de monitoramento e integração sistêmica.

Em suma, a operação do veículo é o resultado da interação contínua destes múltiplos laços de controle. Mensagens de alta frequência e prioridade garantem a responsividade e a segurança, enquanto dados de telemetria e status, com prioridades mais baixas e períodos mais longos, fornecem o *feedback* necessário ao sistema sem comprometer a largura de banda para as funções críticas.

4.4 A ARQUITETURA DE *SOFTWARE*

Se a arquitetura de *hardware* é o corpo do sistema CAN-ESP e o modelo de comunicação é a sua linguagem, a arquitetura de *software* é a sua inteligência. É o *software* que traduz a teoria e o *hardware* em um sistema funcional. Esta seção disseca o principal componente de *software* desenvolvido neste trabalho: a biblioteca `can_esp_lib`, que serve como o núcleo operacional para todas as ECUs da rede.

4.4.1 A Biblioteca de *Software*: `can_esp_lib`

Toda a lógica de comunicação e gerenciamento da rede CAN-ESP é encapsulada em uma biblioteca de *software* customizada, a `can_esp_lib`, desenvolvida como parte central deste trabalho. A criação de uma biblioteca dedicada não é apenas um detalhe de implementação, mas uma decisão estratégica com três objetivos primários: abstração, robustez e reprodutibilidade.

A arquitetura interna da `can_esp_lib` foi projetada de forma modular para cumprir estes três objetivos. Conforme ilustrado no diagrama da Figura 4.5, a biblioteca se posiciona como uma camada intermediária entre a aplicação de alto nível (aplicativo embarcado na ECU) e o *driver* de *hardware* de baixo nível (controlador TWAI), orquestrando o fluxo de dados através de seus componentes principais. A seguir, cada pilar é detalhado em relação a esta arquitetura.

O primeiro pilar, a abstração, é a tradução da complexidade do *hardware* em simplicidade de *software*, um princípio fundamental para o desenvolvimento em sistemas embarcados. Interagir diretamente com o controlador TWAI da ESP-IDF exigiria que a lógica de cada ECU gerenciasse múltiplas estruturas de con-

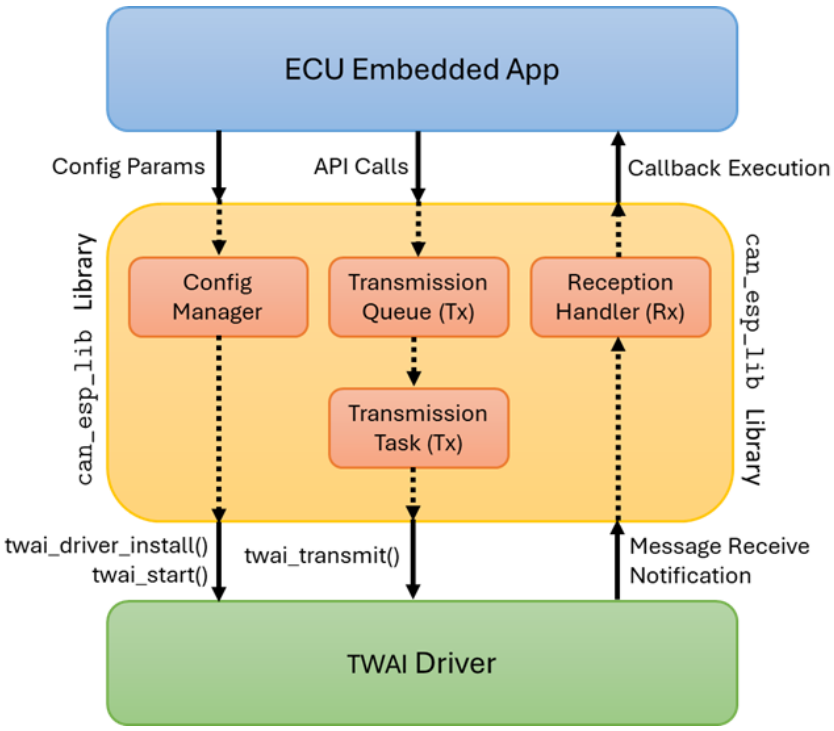

Figura 4.5: Diagrama de Blocos da Arquitetura de Software (biblioteca `can_esp_lib`). Fonte: elaborada pelo autor (2025).

figuração de baixo nível — como as de temporização (`twai_timing_config_t`), de modo de operação (`twai_general_config_t`) e de filtragem de mensagens (`twai_filter_config_t`) — para cada interação com o barramento [25]. A `can_esp_lib` encapsula toda essa complexidade em uma API de alto nível. Com uma única chamada à função `CAN_ESP_InitWithConfig()`, a biblioteca orquestra toda a configuração e inicialização do *hardware*. Similarmente, a função `CAN_ESP_EnqueueMessage()` permite que a aplicação delegue a tarefa de transmissão, com toda a sua lógica de temporização e gerenciamento de *buffers*, a uma tarefa de *background*, liberando o processador para suas funções de controle primárias. Essencialmente, a biblioteca permite que o desenvolvedor se concentre na lógica veicular (o "o quê") em vez de se perder nos detalhes da camada de enlace de dados (o "como").

O segundo pilar, a robustez, é alcançado através de um *design* de *software* que desacopla a lógica da aplicação das incertezas da camada física. Em uma rede CAN, o barramento pode estar temporariamente ocupado por mensagens de maior prioridade e, adicionalmente, podem ocorrer falhas na transmissão que levam a tentativas adicionais de envio no nível do barramento, conforme o mecanismo nativo do protocolo [7, 18]. Se a aplicação tentasse enviar uma mensagem de forma síncrona (ou seja, esperando a transmissão ser concluída), uma tarefa crítica, como a leitura de um sensor, poderia ficar bloqueada, comprometendo o comportamento em tempo real do sistema. A `can_esp_lib` busca mitigar este risco implementando um padrão produtor-consumidor: a aplicação (produtora) simplesmente deposita a mensagem na Fila de Transmissão através de `CAN_ESP_EnqueueMessage()` e continua sua execução imediatamente. Uma tarefa de *background* (consumidora) é a única responsável por gerenciar o envio da mensagem no nível do *driver* — por exemplo, controle de fila, temporização, detecção de falhas de envio e políticas de retentativa no nível da aplicação, quando aplicável —, preservando a responsividade da aplicação principal e permitindo instrumentação/observabilidade do estado do controlador e do barramento.

O terceiro e último pilar, a reprodutibilidade, é garantido pela própria natureza da biblioteca. Ao fornecer uma API única para todas as ECUs da rede, a `can_esp_lib` busca assegurar que o comportamento da camada de comunicação seja idêntico em todos os nós. Isso elimina a possibilidade de erros de implementação inconsistentes entre diferentes ECUs e facilita enormemente a adição de novos nós à rede. Um novo dispositivo, para ser integrado ao sistema, precisa apenas incluir a biblioteca e implementar suas funções de aplicação, herdando automaticamente todo o comportamento de comunicação já validado, o que torna o sistema escalável e de fácil manutenção.

4.5 SOLUÇÃO DE ATUALIZAÇÃO OTA: COMBINANDO ESP-WIFI-MESH, MQTT E HTTPS

A capacidade de atualização de *firmware* OTA é a funcionalidade que eleva o CAN-ESP de uma rede intraveicular "convencional" para uma plataforma alinhada ao paradigma do Veículo Definido por *Software* (SDV). Como demonstrado na análise do estado da arte, esta é uma capacidade crucial ausente até mesmo em sistemas de alto custo, e sua implementação em uma arquitetura de baixo custo representa uma das principais inovações deste trabalho. Esta seção detalha a implementação da arquitetura híbrida para atualização OTA no UGV CELINA, fundamentada na combinação sinérgica de três tecnologias chave: a rede em malha ESP-WIFI-MESH para comunicação local, o protocolo MQTT (*Message Queuing Telemetry Transport*) para sinalização e o protocolo HTTPS (*Hypertext Transfer Protocol Secure*) para a transferência de dados. A arquitetura aqui descrita visa demonstrar uma solução escalável e resiliente, cujas decisões de projeto foram validadas por uma análise do estado da arte e implementadas em uma plataforma de *hardware* real.

4.5.1 Arquitetura de Sistema e Topologia de Rede

A arquitetura global do sistema OTA implementado é visualizada na Figura 4.6, ilustrando a comunicação híbrida e a topologia da rede intraveicular. A solução abrange desde a infraestrutura de nuvem, responsável pelo gerenciamento e distribuição dos *firmwares*, até a rede intraveicular do UGV CELINA. A figura ilustra os três domínios principais da comunicação: a interação entre os servidores de *back-end* (MQTT e HTTPS) e a rede externa; o papel da ECU de Monitoramento e *Gateway* OTA como única ponte de comunicação; e a topologia da rede ESP-WIFI-MESH interna, que interconecta todas as ECUs do veículo.

Como detalhado em seções anteriores, a plataforma experimental para implementação OTA consiste em uma rede CAN com cinco ECUs. Uma ECU foi designada como ECU de Monitoramento e *Gateway* OTA, responsável pela orquestração central, enquanto as outras quatro gerenciam subsistemas específicos (tração, direção, sensores, etc.).

A comunicação interna entre as ECUs é implementada através do protocolo ESP-WIFI-MESH. A escolha desta tecnologia em detrimento de uma rede Wi-Fi de infraestrutura tradicional foi estratégica. O ESP-WIFI-MESH oferece uma topologia de rede em malha auto-organizável e autorreparável, o que confere alta resiliência a falhas de nós individuais e a variações na qualidade do *link* de rádio, cenário comum em um ambiente operacional de um veículo. Essa capacidade de autorreparação, fundamentada

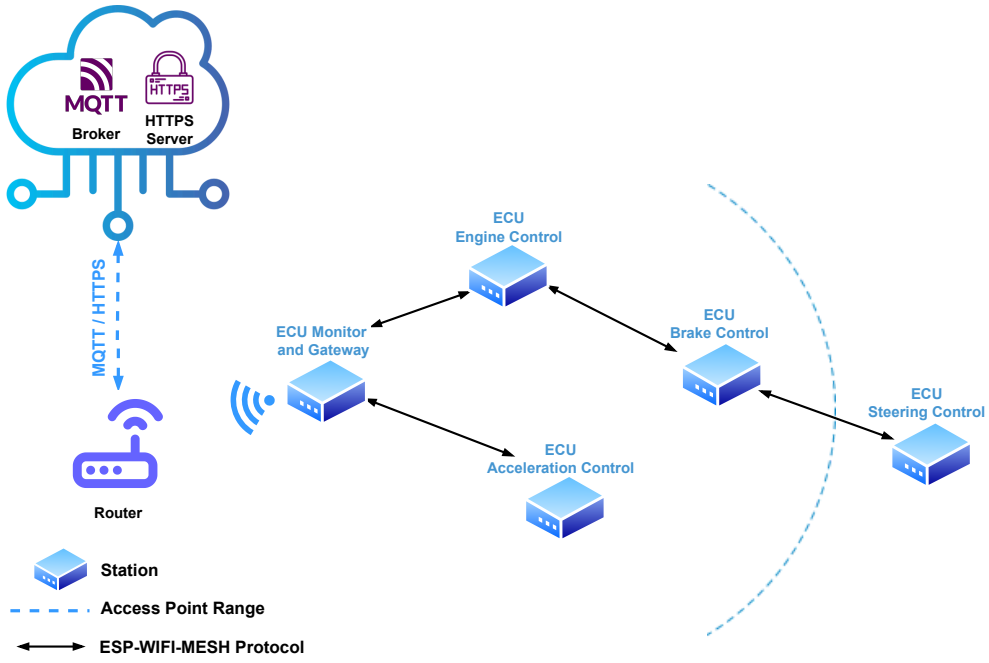

Figura 4.6: Arquitetura global do sistema de atualização OTA. Fonte: elaborada pelo autor (2025).

na redundância de rotas da topologia *mesh*, representa uma vantagem fundamental sobre uma arquitetura Wi-Fi convencional, na qual a falha do ponto de acesso centralizado representa um ponto único de falha (*single point of failure*) que comprometeria toda a operação de gerenciamento remoto [44].

Nesta topologia, a ECU de Monitoramento e *Gateway* OTA opera como o nó raiz (*root node*), sendo o único elemento da rede veicular com conexão a uma rede Wi-Fi externa. As demais ECUs funcionam como nós filhos (*child nodes*), restringindo sua comunicação à malha interna. Esta configuração cria um perímetro de segurança natural, minimizando a superfície de ataque ao expor apenas um ponto de acesso à uma rede externa.

4.5.2 Arquitetura de Comunicação Híbrida com a Nuvem

Para a comunicação com os servidores de *back-end*, foi adotado um modelo híbrido que segrega o Plano de Controle do Plano de Dados. A literatura aponta que, embora protocolos como MQTT sejam eficientes para mensagens de telemetria e comando, a transferência de arquivos binários de tamanho maior pode sobrecarregar o servidor (*broker*) e introduzir complexidade desnecessária. Uma arquitetura que utiliza protocolos distintos para sinalização e para transferência de dados massivos é reconhecida como mais escalável e robusta, sendo uma abordagem de projeto recomendada para sistemas FOTA (*Firmware Over-The-Air*) flexíveis [45].

4.5.2.1 Plano de Controle via MQTT

O protocolo MQTT foi selecionado para o Plano de Controle devido às suas características de leveza, baixo *overhead* e modelo de publicação/assinatura, ideal para a troca de mensagens de comando e status de baixa latência [46]. Seu uso em sistemas de telemetria e controle veicular é bem estabelecido, dadas as condições de conectividade frequentemente intermitentes desses ambientes [47]. Para implementar este plano, foi definida uma estrutura de tópicos específica para segregar os fluxos de comando e de *feedback*, conforme detalhado na Tabela 4.3.

Tabela 4.3: Estrutura de Tópicos MQTT

Tipo	**Tópico MQTT**	**Descrição da Função**
Comando (*Downstream*)	`ota/jobs/{ID_VEICULO}`	Tópico utilizado exclusivamente pelo servidor de *back-end* para publicar ordens de atualização direcionadas a um veículo específico. A ECU de Monitoramento e *Gateway* OTA de cada veículo subscreve a este tópico usando seu ID único. O *payload* da mensagem é um objeto JSON contendo todos os metadados necessários para a ECU iniciar o download via HTTPS.
Feedback (*Upstream*)	`ota/status/{ID_VEICULO}/{ID_ECU}`	Tópico utilizado pela ECU de Monitoramento e *Gateway* OTA para publicar mensagens de status sobre o progresso da atualização de uma ECU específica dentro do veículo. O servidor de *back-end* subscreve a um *wildcard* (e.g., `ota/status/+/+`) para monitorar toda a frota. Permite o rastreamento granular de cada etapa do processo OTA.

4.5.2.2 Plano de Dados via HTTPS

O processo de transferência do arquivo binário do *firmware* é delegada ao protocolo HTTPS. A ECU de Monitoramento e *Gateway* OTA, ao receber a ordem via MQTT, utiliza um cliente HTTPS para realizar o download a partir da URL especificada. Esta abordagem alavanca a infraestrutura padrão da web, otimizada para a distribuição de conteúdo de grande volume, e garante a confidencialidade e integridade da transferência através do protocolo TLS (*Transport Layer Security*). O uso de um canal seguro como o HTTPS é reconhecido como uma contramedida essencial para proteger o processo de atualização OTA contra ataques de interceptação (Man-in-the-Middle) [48]. Além disso, seu uso para o plano de dados, separadamente do plano de controle, é uma prática que confere maior escalabilidade e flexibilidade a sistemas de atualização de *firmware* para IoT [45].

4.5.3 Orquestração do Processo de Atualização

A orquestração do processo de atualização OTA compreende a sequência de operações lógicas executadas pela ECU de Monitoramento e *Gateway* OTA para garantir a transferência segura e confiável de um novo *firmware* desde a infraestrutura de nuvem até uma ECU de destino na rede CAN-ESP. Este fluxo

é iniciado a partir do recebimento de uma ordem de atualização, publicada pelo servidor de *back-end* no tópico MQTT de "Comando".

A partir deste gatilho, a ECU de Monitoramento e *Gateway* OTA assume a responsabilidade por coordenar uma série de etapas interdependentes: o download do binário de *firmware* via HTTPS, seu armazenamento intermediário em um dispositivo de memória não volátil, a distribuição fragmentada para o nó de destino através da rede *mesh* e a supervisão do ciclo de *feedback*.

Para garantir a segurança operacional e evitar a sobrecarga do barramento durante a condução crítica, o protocolo de orquestração implementa uma verificação de "estado seguro" (*Safe State Check*). O processo de atualização OTA é estritamente condicionado a dois pré-requisitos lógicos: (i) o veículo deve estar com a ignição desligada (*Vehicle-Off status*), garantindo que o trem de força e os sistemas de direção estejam em estado de repouso (*Bus Idle*); e, (ii) a confirmação explícita do usuário deve ser obtida via Interface Homem-Máquina (HMI). Esta estratégia busca garantir que a largura de banda da rede *mesh* e do barramento CAN seja dedicada exclusivamente à transferência de *firmware* apenas quando não há competição com mensagens de controle de tempo real, prevenindo estresse desnecessário nas ECUs e eliminando riscos de segurança viária.

As subseções a seguir detalham cada uma dessas fases, elucidando os protocolos e mecanismos implementados em cada etapa.

4.5.3.1 Armazenamento Intermediário no Nó *Gateway*

A RAM do ESP32 é insuficiente para armazenar o arquivo de *firmware* completo, exigindo um mecanismo de *store-and-forward*. Dado que a ECU de Monitoramento e *Gateway* OTA já possui um módulo de cartão MicroSD para o *logging* de dados da rede, este foi utilizado como armazenamento intermediário. Esta abordagem sinérgica reutiliza o *hardware* existente e, crucialmente, preserva os ciclos de escrita da memória *flash* interna do microcontrolador, aumentando a longevidade do dispositivo. Os *firmwares* baixados são armazenados em um diretório dedicado (`/ota_updates/`) no cartão SD.

4.5.3.2 Protocolo de Distribuição na Rede *Mesh*

A distribuição do *firmware* da ECU de Monitoramento e *Gateway* OTA para a ECU de destino através da rede *mesh* requer a implementação de um protocolo na camada de aplicação, projetado para gerenciar a fragmentação do arquivo e garantir a confiabilidade da transferência. Este protocolo opera em um modelo cliente-servidor, onde a ECU de Monitoramento e *Gateway* OTA atua como servidor e a ECU de destino como cliente. O fluxo de mensagens que compõe este protocolo de transferência, desde o *handshake* inicial até a finalização, é visualizado, em detalhe, no diagrama de sequência da Figura 4.7.

As etapas desta comunicação são detalhadas a seguir:

1. **Handshake:** a ECU de Monitoramento e *Gateway* OTA (ECU de origem) envia uma mensagem de iniciação para a ECU a ser atualizada (ECU de destino) com os metadados do *firmware*.
2. **Confirmação:** a ECU de destino responde com um sinal de prontidão.

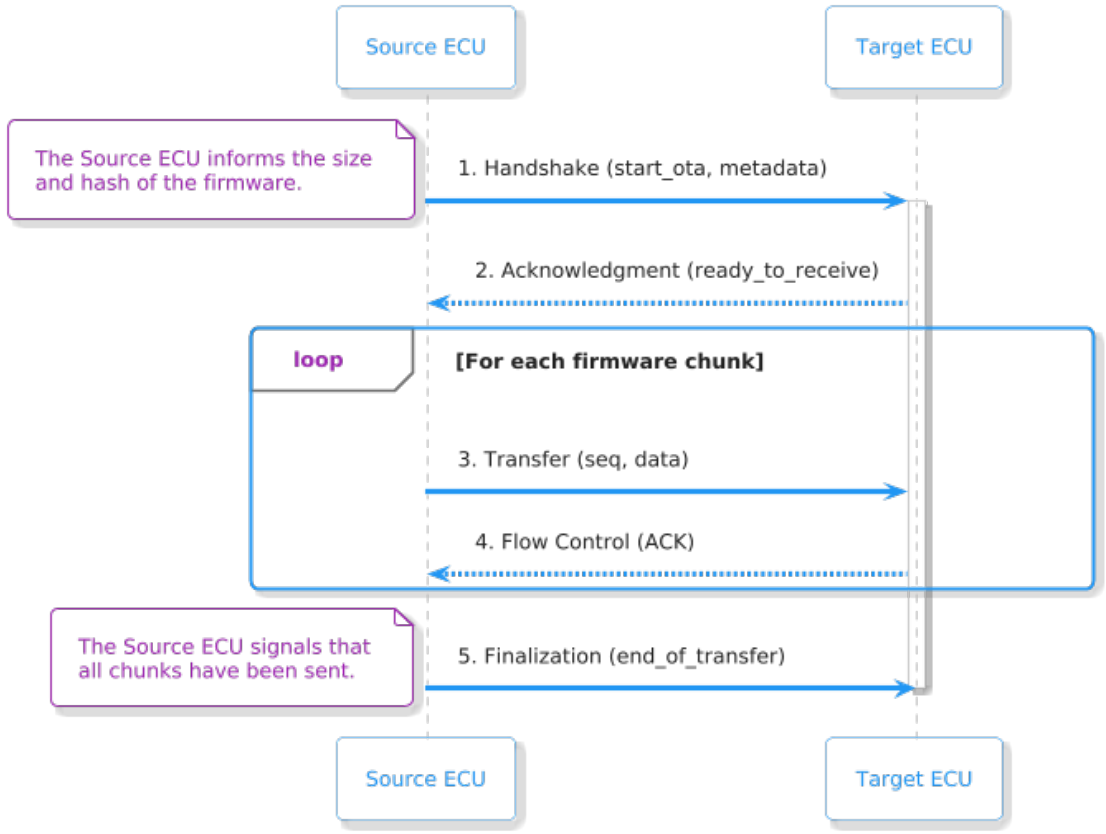

Figura 4.7: Diagrama de sequência do protocolo de transferência de *firmware* na rede ESP-WIFI-MESH. Fonte: elaborada pelo autor (2025).

3. **Transferência de *Chunks*:** a ECU de origem lê o arquivo do cartão SD em pedaços de 1024 bytes, encapsula-os com um número de sequência e os envia via `esp_mesh_send()`.

4. **Controle de Fluxo:** a ECU de destino envia uma confirmação (*Acknowledgement* - ACK) a cada N pacotes, para evitar a sobrecarga de seus *buffers* de recepção. Este mecanismo de controle de fluxo é crucial para a robustez da transferência em um sistema embarcado com recursos de memória limitados, pois previne a ocorrência de *buffer overflow* na ECU receptora, uma condição que poderia levar não apenas à perda de pacotes, mas também à instabilidade do sistema ou a vulnerabilidades de segurança [49].

5. **Finalização:** após o último *chunk*, a ECU de origem envia uma mensagem de finalização.

4.5.3.3 Processo de Escrita e Ativação na ECU de Destino

Uma vez que o protocolo de distribuição entrega os dados à ECU de destino, a lógica de *software* interna assume o controle para executar a atualização de forma segura. Este processo foi modelado como uma máquina de estados finitos, garantindo que a ECU opere de maneira previsível e seja capaz de se recuperar de falhas. O diagrama de máquina de estados, apresentado na Figura 4.8, ilustra o ciclo de vida completo da atualização na ECU de destino, desde o recebimento do comando inicial até a reinicialização com o novo *firmware*.

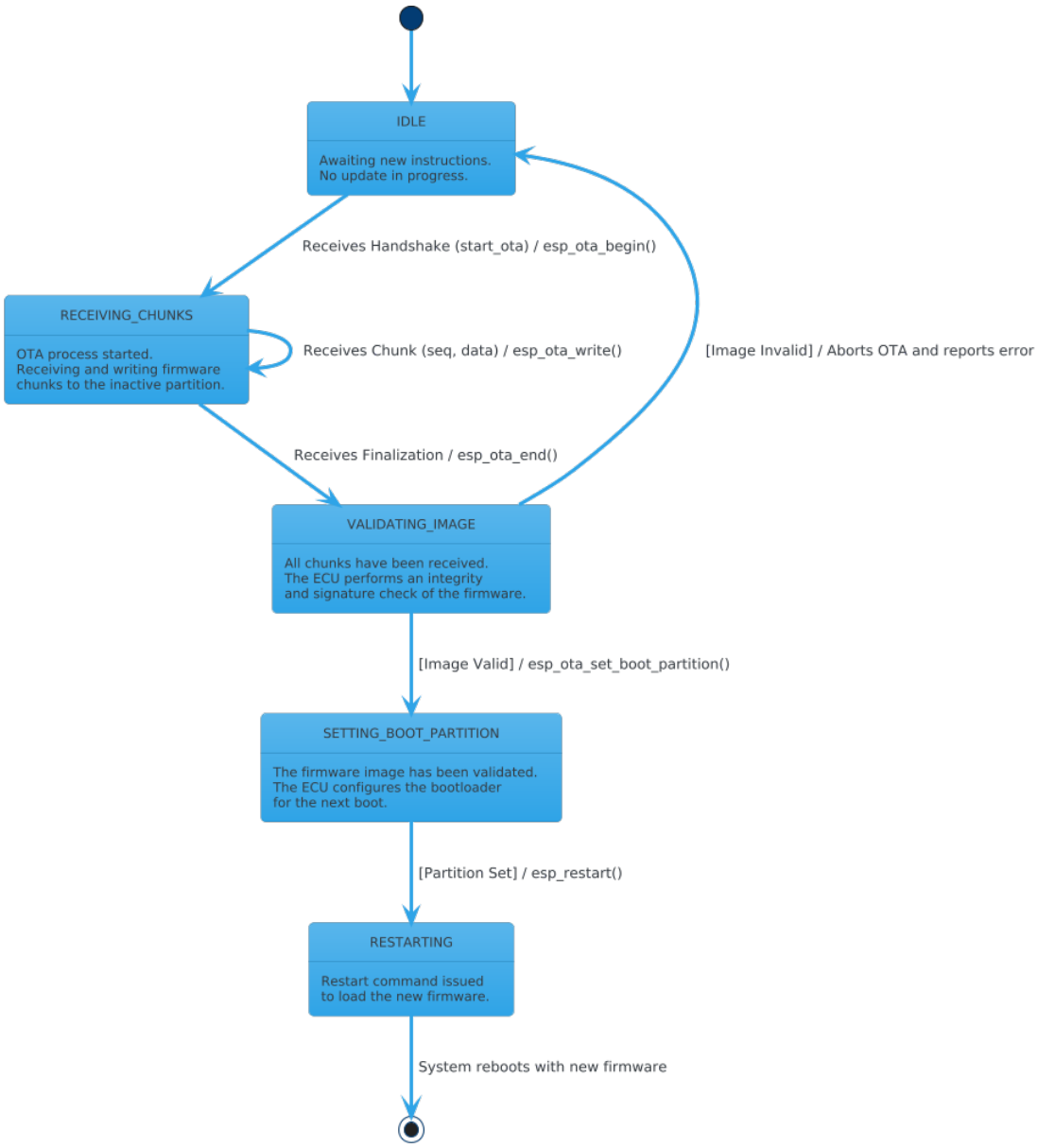

Figura 4.8: Diagrama de máquina de estados do processo OTA na ECU de destino. Fonte: elaborada pelo autor (2025).

Conforme ilustrado, a ECU permanece em estado OCIOSO até receber o comando de *handshake* do *gateway*. Neste momento, a função `esp_ota_begin()` é chamada para preparar a partição de destino, e a ECU transita para o estado RECEBENDO_CHUNKS. Neste estado, ela permanece em um laço, recebendo cada *chunk* de *firmware* e gravando-o sequencialmente na partição inativa através de chamadas sucessivas a `esp_ota_write()`. Ao receber o sinal de finalização da transferência, a transição para o estado VALIDANDO_IMAGEM é acionada pela chamada a `esp_ota_end()`, que finaliza a escrita e executa as verificações de consistência e integridade da imagem (por exemplo, estrutura/formato e checagens internas da imagem). A verificação criptográfica de autenticidade por assinatura digital, quando habilitada na plataforma, é tipicamente aplicada no processo de inicialização pelo *bootloader*, como parte do *hardening* do mecanismo OTA. Estes recursos de segurança são nativos do *framework* ESP-IDF.

Neste ponto crítico, o fluxo se bifurca: se a imagem for considerada inválida, o processo é abortado e a ECU retorna de forma segura ao estado OCIOSO, reportando o erro. Se a imagem for válida, a função `esp_ota_set_boot_partition()` é executada, movendo a máquina para o estado ATIVANDO_PARTICAO. Finalmente, a chamada a `esp_restart()` transiciona a ECU para o estado REINICIANDO, completando o ciclo para carregar o novo *firmware* na próxima inicialização.

4.5.4 Implementação de Segurança e Confiabilidade

A viabilidade de um sistema de atualização OTA em um ambiente de produção, especialmente no domínio veicular, está diretamente relacionada com a robustez de seus mecanismos de segurança e confiabilidade. Para além da funcionalidade da transferência de dados, é imperativo garantir que o processo seja resiliente a falhas que poderiam tornar um dispositivo inoperante e mantê-lo protegido contra vetores de ataque que visam comprometer a integridade do sistema. Nesse sentido, a arquitetura implementada adota uma abordagem de defesa em profundidade, utilizando funcionalidades nativas do ESP-IDF para mitigar riscos específicos, abordando a resiliência operacional, a autenticidade do *firmware* e a prevenção contra ataques de *downgrade*.

A primeira salvaguarda implementada foca na resiliência operacional através do mecanismo de *rollback* nativo da plataforma embarcada. Com este mecanismo, a aplicação recém-instalada deve obrigatoriamente se autovalidar, invocando a função `esp_ota_mark_app_valid_cancel_rollback()` somente após uma rotina de autodiagnóstico ser concluída com sucesso. Caso a aplicação falhe, trave ou reinicie antes de realizar esta chamada, o *bootloader* reverterá automaticamente para a imagem funcional anterior. Esta capacidade de *rollback* é um mecanismo de resiliência crítico, projetado para mitigar o risco de *bricking* do dispositivo (a inutilização do *hardware* por uma atualização corrompida ou incompatível), uma das falhas mais catastróficas em processos OTA [50]. Complementando a resiliência, a segurança do processo pode ser reforçada pela verificação criptográfica de autenticidade do *firmware*, quando habilitadas as funcionalidades de segurança da plataforma (por exemplo, mecanismos de inicialização segura e validação de imagem). Nessa configuração, o *bootloader* verifica a integridade/autenticidade do binário antes de permitir sua execução, reduzindo o risco de instalação de *software* não autorizado e constituindo uma defesa primária contra vetores de ataque em sistemas conectados [50].

Finalmente, para proteger contra o possível vetor de ataque de *downgrade*, no qual se tenta reinstalar uma versão de *firmware* antiga e vulnerável, a arquitetura contempla o uso de um mecanismo de *anti-rollback*. Esta funcionalidade utiliza *eFuses* no chip para armazenar uma versão de segurança que só pode ser incrementada, uma técnica de *hardening* recomendada em regulações de cibersegurança automotiva como a UNECE UN R155 [51].

4.5.5 O Ciclo de *Feedback* de Malha Fechada

A robustez da arquitetura proposta não deriva apenas da conectividade do *hardware*, mas do protocolo de monitoramento de estados de malha fechada desenhado neste trabalho. Diferente de abordagens do tipo "dispare e esqueça" (*fire-and-forget*), que carecem de visibilidade sobre o resultado da operação, o sistema implementa uma verificação determinística em cada etapa crítica (download, distribuição, valida-

ção), conforme detalhado na Tabela 4.4. Este mecanismo constitui a principal contribuição lógica para a confiabilidade do sistema SDV de baixo custo proposto. Tal observabilidade é um requisito não funcional crítico para sistemas de produção, pois permite a detecção de falhas, a automação de novas tentativas e o monitoramento preciso do estado das ECUs dos veículos. Para concretizar essa visão, a arquitetura implementa um ciclo de *feedback* em duas etapas distintas, transportando a informação de estado desde a ECU de destino até a plataforma de *back-end*.

A primeira etapa, denominada *feedback* local, ocorre inteiramente na rede *mesh*. Após cada marco significativo do processo de atualização, a ECU de destino (e.g., "ECU de Controle do Freio") compõe uma mensagem de status estruturada e a envia para a ECU de Monitoramento e *Gateway* OTA através da rede *mesh*. Estas mensagens informam sobre o progresso, como o recebimento completo do arquivo, a verificação bem-sucedida de sua integridade, ou o sucesso final da instalação após a reinicialização e autovalidação. Em caso de falha, um código de erro específico é reportado, permitindo um diagnóstico preciso da causa raiz.

Subsequentemente, na segunda etapa, a ECU de Monitoramento e *Gateway* OTA atua como uma ponte de comunicação, realizando o *feedback* remoto. Ao receber um relatório de status da rede *mesh*, a ECU de Monitoramento e *Gateway* OTA o enriquece com metadados adicionais, adicionando um *timestamp*, e o retransmite para o tópico MQTT de status correspondente. Esta retransmissão não é apenas um encaminhamento passivo; a própria ECU funciona como um *proxy* de estado, garantindo que a comunicação com a nuvem seja eficiente e que a informação de status seja contextualizada.

A granularidade deste ciclo de *feedback* permite que a plataforma de *back-end* construa uma visão detalhada do processo, tratando a atualização de cada ECU como uma máquina de estados finitos. A tabela 4.4 detalha os principais estados reportados durante o ciclo de vida de uma atualização OTA, a origem de cada reporte e sua respectiva descrição.

Tabela 4.4: Estados do Ciclo de *Feedback* OTA

Estado	**Origem do Reporte**	**Descrição da Função**
`PENDING`	Servidor de *Back-end*	O trabalho de atualização foi criado e publicado no tópico MQTT, mas ainda não foi processado pela ECU de Monitoramento e *Gateway* OTA.
`DOWNLOADING`	ECU de Monitoramento e *Gateway* OTA	A ECU de Monitoramento e *Gateway* OTA iniciou o download do arquivo de *firmware* a partir do servidor HTTPS.
`DISTRIBUTING`	ECU de Monitoramento e *Gateway* OTA	O download foi concluído com sucesso e a ECU de Monitoramento e *Gateway* OTA iniciou a distribuição do *firmware* para a ECU de destino via rede *mesh*.
`INSTALLING`	ECU de Destino	A ECU de destino recebeu o arquivo, validou seu *hash* e iniciou o processo de escrita na partição OTA inativa.
`AWAITING_VALIDATION`	ECU de Destino	A escrita foi concluída e a ECU de destino reiniciou com o novo *firmware*. Encontra-se no período de autodiagnóstico.
`SUCCESS`	ECU de Destino	O autodiagnóstico foi bem-sucedido, a aplicação foi marcada como válida e a atualização está permanentemente concluída.
`FAILED_DOWNLOAD`	ECU de Monitoramento e *Gateway* OTA	A ECU de Monitoramento e *Gateway* OTA falhou ao tentar baixar o arquivo do servidor HTTPS (e.g., erro de rede, arquivo não encontrado).
`FAILED_VALIDATION`	ECU de Destino	O autodiagnóstico da nova aplicação falhou, e o mecanismo de *rollback* foi acionado, revertendo para a versão anterior.

Esta visão de estado detalhada habilita funcionalidades avançadas de gerenciamento. Por exemplo, uma falha com o estado `FAILED_DOWNLOAD` pode automaticamente agendar uma nova tentativa, enquanto uma falha recorrente do tipo `FAILED_VALIDATION` em uma ECU específica pode gerar um alerta para intervenção humana. Na prática, este ciclo de *feedback* transforma a atualização de *firmware* de uma operação de "disparar e esquecer" (*fire-and-forget*) em um processo de gerenciamento do ciclo de vida do *software* distribuído totalmente observável. Isso permite que o estado de cada ECU na rede seja monitorado de forma centralizada, um requisito fundamental para a manutenção, a segurança e a operação escalável de sistemas IoT complexos [50].

4.5.6 Modelagem Teórica de Escalabilidade

A viabilidade da arquitetura CAN-ESP em veículos de maior complexidade depende diretamente da escalabilidade do protocolo de distribuição de *firmware* na rede *mesh*. Considerando uma topologia em cadeia (pior caso para latência), o tempo total para propagação de uma atualização (T_{total}) pode ser modelado em função do número de saltos (h) e do tamanho do *firmware* (S).

O rendimento efetivo (η) da rede *mesh* sofre degradação não-linear com o aumento da profundidade da malha devido à contenção do meio compartilhado e ao *overhead* de retransmissão, um comportamento característico de redes *ad-hoc* sem fio [44]. Baseado no modelo de capacidade de redes *multi-hop* [52], o tempo de propagação para o nó mais distante em uma rede de profundidade h é aproximado por:

$$T_{total}(h, S) = \frac{S}{\eta_{base}} \times (1 + \alpha \cdot (h - 1)) + T_{overhead} \cdot h \tag{4.1}$$

Onde η_{base} é o *throughput* efetivo no primeiro salto (ordem de grandeza reportada para o protocolo ESP-WIFI-MESH [28]), α é um coeficiente de degradação por salto que agrega efeitos de contenção do meio, *backoff* do CSMA/CA e retransmissões, e $T_{overhead}$ representa o tempo de processamento de *handshake* em cada nó. Ressalta-se que esses parâmetros são dependentes do cenário (interferência, topologia, densidade de nós, potência/qualidade do enlace e implementação), de modo que o modelo deve ser interpretado como uma aproximação de primeira ordem para análise de tendência e dimensionamento.

Para um cenário hipotético de um veículo comercial pesado com 50 ECUs (organizadas em uma malha com profundidade máxima $h = 10$), e um *firmware* típico de 1 MB ($S = 8 \cdot 10^6$ bits), a estimativa resultante situa o tempo de propagação total na ordem de dezenas de segundos a poucos minutos, a depender dos parâmetros efetivos do enlace e do *overhead* de controle. Mesmo sob degradação inerente a redes *multi-hop* [44], tal ordem de grandeza permanece compatível com janelas usuais de manutenção (minutos ou horas), sustentando a viabilidade prática do mecanismo de distribuição para frotas internas de média densidade, desde que respeitadas as premissas operacionais de execução em estado seguro do veículo.

Em síntese, essa modelagem reforça o argumento de escalabilidade do subsistema OTA ao mostrar que a segregação dos planos de controle e dados, combinada com a distribuição na malha local, pode atender a requisitos operacionais de atualização em janelas de manutenção, preservando a lógica de observabilidade e controle de estado descrita nas seções anteriores.

5 AVALIAÇÃO DE DESEMPENHO

Este capítulo apresenta a avaliação quantitativa de desempenho do CAN-ESP, com foco em métricas que caracterizam eficiência de uso do barramento, robustez e propriedades de tempo real em redes CAN. Em particular, são avaliadas a carga do barramento, a latência de comunicação (via *Event Time Stamping*), a taxa de retransmissão, os contadores de erro e a integridade da comunicação, o tempo de resposta do sistema, o *jitter* e o pior caso de tempo de resposta (WCRT), tomando como referência a especificação CAN 2.0, a norma ISO para CAN e métodos clássicos de análise de tempo de resposta e escalonabilidade em redes CAN [7, 18, 24, 53, 12, 6].

5.1 METODOLOGIA EXPERIMENTAL

5.1.1 Premissas experimentais e separação entre avaliação CAN e atualização OTA

Os ensaios deste capítulo são estritamente dedicados à caracterização do comportamento do barramento CAN no modo operacional do veículo, isto é, durante a execução das funções de controle, telemetria e diagnóstico distribuído suportadas pela arquitetura CAN-ESP. A atualização remota de *firmware* OTA não é tratada como fonte de tráfego no barramento CAN, uma vez que todo o processo de distribuição do binário entre as ECUs ocorre na infraestrutura sem fio, por meio da rede *mesh* [28]. Além disso, por decisão de projeto, o procedimento de atualização é executado apenas com o veículo desligado e mediante confirmação explícita do usuário, evitando gerar interferências com requisitos de tempo real durante a condução. Consequentemente, as seções referentes à análise das métricas propostas neste trabalho (Seções 5.2.1–5.2.8) devem ser interpretadas como propriedades do barramento CAN sob tráfego operacional.

5.1.2 Plataforma experimental e instrumentação

A avaliação de desempenho foi conduzida sobre a plataforma veicular do UGV CELINA, previamente apresentada no Capítulo 4, com ênfase nos elementos de *hardware* e ferramentas de análise descritos na Seção 4.2. Dessa forma, a presente seção descreve apenas os aspectos necessários para a reprodutibilidade dos experimentos de desempenho, evitando redundância com as especificações já consolidadas no texto.

A instrumentação adotada combina (i) a captura do tráfego no barramento CAN por meio de interface de monitoramento, e (ii) a inspeção em baixo nível do sinal elétrico com analisador lógico, conforme as ferramentas de análise indicadas na Tabela 4.1. Adicionalmente, métricas relacionadas ao estado do controlador (por exemplo, contadores de erro e estado do barramento) são obtidas por meio das estruturas de *status* disponibilizadas por funções disponíveis na API do controlador TWAI no ESP-IDF [25]. A configuração do barramento (incluindo terminação e parâmetros de camada física) segue as recomendações do protocolo CAN e da norma ISO 11898 [7, 18].

5.1.3 Cenários experimentais e níveis de carga

Para manter o esforço total controlado e, ao mesmo tempo, garantir poder discriminativo na análise, os ensaios foram organizados em quatro cenários operacionais, combinados com um conjunto enxuto de níveis-alvo de carga do barramento. No cenário de referência (*baseline*), a carga foi varrida em quatro patamares (leve, moderada, alta e limite prático) para produzir tendências em função da ocupação do barramento. Nos demais cenários, a avaliação foi restrita aos patamares mais informativos (altas cargas), os quais tendem a maximizar o impacto de competição por arbitragem sobre latência, *jitter* e WCRT em redes CAN priorizadas por identificador [24, 53]. Adicionalmente, o cenário S4 foi definido como uma condição de estresse máximo na plataforma, em que a carga do barramento é intencionalmente sustentada acima de 90% por tráfego artificial gerado na rede CAN, com o objetivo de caracterizar o comportamento em saturação forçada.

Tabela 5.1: Matriz enxuta de cenários e níveis-alvo de carga do barramento utilizados na avaliação.

Cenário	**Níveis-alvo de carga do barramento**
S1 – *Baseline* (tráfego operacional típico)	20%, 40%, 60%, 80%
S2 – Competição de arbitragem (tráfego *background* de baixa prioridade)	60% a < 90%
S3 – Escalabilidade (aumento de nós/fluxos mantendo classes de mensagens)	40% a 60%
S4 – Robustez/Integridade (condição de estresse na plataforma)	> 90%

5.1.4 Tamanho amostral e repetições por condição experimental

Para fins de análise, consideramos um tamanho médio de frame de 128 bits, que corresponde à média ponderada dos diferentes DLCs na rede. Embora mensagens individuais possam variar (ex: mensagens com DLC=1 têm 66 bits, com DLC=8 têm 174 bits), o valor médio de 128 bits fornece uma aproximação válida para análise agregada do barramento.

Cada condição experimental (par cenário–carga) foi executada até a troca de $3\,000\,000$ quadros de dados entre as ECUs, critério adotado para assegurar uma amostragem extensa e estabilidade estatística na estimação de métricas de latência, *jitter*, retransmissões e integridade. Para capturar variações entre execuções e garantir reprodutibilidade, cada condição foi repetida de forma independente por $R = 3$ repetições, resultando em um total de $9\,000\,000$ quadros por condição. As repetições foram realizadas com reinicialização controlada do sistema para reduzir dependência de estado residual (por exemplo, filas de transmissão e estado interno do controlador), e os resultados foram consolidados a partir das distribuições obtidas em cada repetição. Essa estratégia é compatível com avaliações experimentais de comunicação em CAN reportadas na literatura, nas quais se emprega coleta extensiva de tráfego e repetição para caracterização robusta de propriedades temporais [9, 43]. A seguir, apresentam-se os resultados obtidos para as métricas definidas.

5.2 RESULTADOS OBTIDOS

Esta seção apresenta os resultados quantitativos seguindo a organização das métricas definidas na Seção 5.1. Inicia-se pela caracterização da ocupação do barramento e, em seguida, são analisadas métricas temporais (latência, variabilidade da latência e WCRT observado) e de integridade (retransmissão, contadores de erro e perda de arbitragem), com ênfase nos efeitos de competição por arbitragem em diferentes condições de carga [7, 18, 24, 53, 12].

5.2.1 Carga do Barramento

O monitoramento da carga do barramento (*bus load*) durante a transmissão das $3\,000\,000$ mensagens na condição nominal evidenciou variações na faixa de 10% a 25%, caracterizando um regime de operação com folga de ocupação do meio no UGV CELINA e indicando margem para evolução do conjunto de mensagens sem saturar o barramento (Figura 5.1). O cálculo da carga foi realizado conforme a Equação 5.1, na qual T_{frame} representa o tempo total de transmissão dos quadros no período de observação e T_{total} corresponde ao tempo total de operação do barramento.

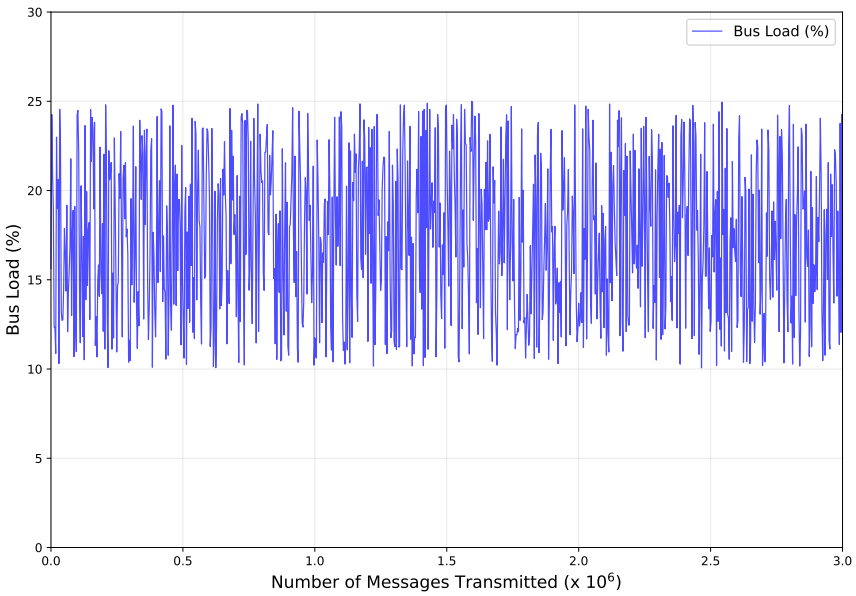

Figura 5.1: Variação da Carga do Barramento CAN (*Bus Load*). Fonte: elaborada pelo autor (2025).

$$\text{Bus Load}(\%) = \frac{\sum T_{\text{frame}}}{T_{\text{total}}} \times 100 \tag{5.1}$$

Os resultados dos experimentos indicaram que o índice de ocupação se manteve abaixo de 25%, o que possibilita a inclusão de novos nós à rede com baixo risco de saturação.

Os valores supracitados (10% a 25%) referem-se ao cenário nominal (S1). Conforme estabelecido nas premissas deste capítulo, o processo de atualização OTA não contribui para a carga do barramento CAN, pois a distribuição do binário entre as ECUs ocorre exclusivamente pela rede *mesh* [28]. No cenário de referência (*baseline*), as métricas temporais analisadas nas subseções seguintes refletem os efeitos combi-

nados de arbitragem e do bloqueio inerente ao CAN, uma vez que o protocolo é não preemptivo e uma mensagem de alta prioridade não interrompe um quadro já em transmissão [7, 18, 24, 53, 12].

5.2.2 Taxa de Retransmissão de Quadros (Retransmission Rate)

A taxa de retransmissão quantifica a incidência de falhas detectadas durante a transmissão de quadros CAN e a consequente necessidade de reenvio automático, mecanismo de robustez previsto na especificação CAN e na norma ISO 11898 [7, 18]. No CAN-ESP, essa métrica é utilizada como indicador do comportamento da camada física e do ambiente eletromagnético durante a operação, sendo obtida a partir da instrumentação em *software* e da inspeção do estado do controlador TWAI (por exemplo, sinais de erro e condições do barramento) através de funções da API disponibilizada no ESP-IDF [25].

Para evitar ambiguidade entre taxa instantânea e agregada, a retransmissão é reportada de duas formas complementares: (i) taxa global média normalizada ao volume total de tráfego e (ii) taxa por janela ao longo do experimento, para evidenciar variações temporais e possíveis rajadas de erros. Como unidade primária, adota-se *ppm* (*parts per million*), isto é, eventos por 10^6 quadros, por ser mais legível para taxas baixas.

Após verificações e correções de camada física (por exemplo, mitigação de ruído/EMI, revisão de conexões e terminação), a taxa média global observada foi da ordem de 15 eventos por 10^6 quadros, isto é, 15 ppm, o que equivale a $0{,}0015\%$ do tráfego total. Para o volume de $3\,000\,000$ quadros, essa ordem de grandeza corresponde aproximadamente a 45 eventos de retransmissão no experimento completo.

Para caracterização temporal, a Figura 5.2 apresenta a evolução da taxa de retransmissão **por janela** ao longo do experimento. Nesta avaliação, cada ponto do gráfico corresponde ao cálculo em uma janela de $W = 100\,000$ quadros transmitidos. Seja $N_{\text{tx}}^{(w)}$ o número total de quadros transmitidos (ou tentativas válidas contabilizadas) dentro da janela w e $N_{\text{retrans}}^{(w)}$ o número de eventos de retransmissão observados nessa mesma janela; a taxa por janela é então calculada conforme as Equações 5.2 e 5.3. A taxa global média é obtida de forma análoga, substituindo-se $N_{\text{tx}}^{(w)}$ e $N_{\text{retrans}}^{(w)}$ pelos totais acumulados ao longo de todo o experimento.

A taxa em *ppm* é definida por:

$$\text{Retransmission Rate (ppm)} = \frac{N_{\text{retrans}}}{N_{\text{tx}}} \times 10^6, \tag{5.2}$$

em que N_{retrans} é o número de eventos de retransmissão observados e N_{tx} é o número total de quadros transmitidos (ou tentativas válidas contabilizadas) no mesmo intervalo de observação (janela ou experimento completo). Como 1 ppm $= 10^{-6}$ em razão, a conversão para porcentagem resulta diretamente em:

$$\text{Retransmission Rate } (\%) = \frac{N_{\text{retrans}}}{N_{\text{tx}}} \times 100 = \frac{\text{Retransmission Rate (ppm)}}{10^4}. \tag{5.3}$$

No contexto experimental considerado, valores na ordem de dezenas de *ppm*, combinados com a ausência de estados críticos do controlador (por exemplo, *Bus-Off*, discutido na Seção 5.2.3), caracterizam uma operação estável do barramento, com incidência de erros suficientemente baixa para não comprometer a integridade da comunicação e as propriedades temporais avaliadas nas demais métricas.

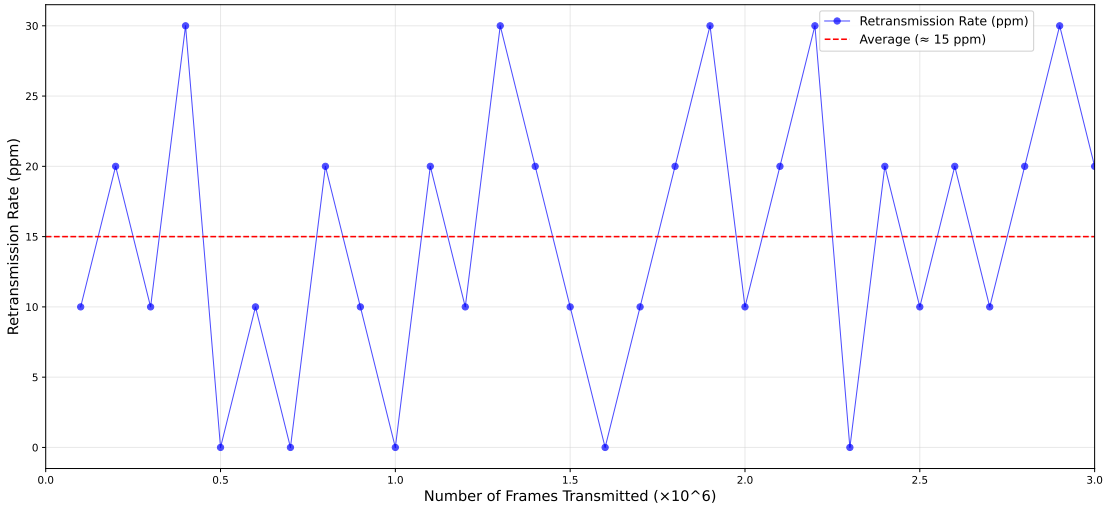

Figura 5.2: Taxa de Retransmissão por janela (*Retransmission Rate*), com $W = 100\,000$ quadros por ponto, reportada em *ppm* (eventos por 10^6 quadros). Fonte: elaborada pelo autor (2025).

5.2.3 Contadores de Erro (*Error Counters*) e Integridade da Comunicação

A integridade da comunicação foi verificada por meio dos contadores de erro do CAN, que quantificam, em cada nó, a incidência de eventos de falha detectados durante transmissão e recepção e sustentam o mecanismo de confinamento de erro (*error confinement*) previsto na especificação CAN e na norma ISO 11898 [7, 18]. Na implementação CAN-ESP, esses valores são obtidos diretamente do controlador TWAI por meio das estruturas de *status* disponibilizadas no ESP-IDF [25].

Cada ECU mantém dois contadores: o Contador de Erros de Transmissão (*Transmit Error Counter* – TEC) e o Contador de Erros de Recepção (*Receive Error Counter* – REC), ajustados conforme o sucesso ou falha na transmissão e recepção de quadros. A partir desses contadores, o estado do nó é classificado como: *Error Active* (operação normal), quando $TEC < 128$ e $REC < 128$; *Error Passive*, quando $128 \leq TEC < 256$ ou $128 \leq REC < 256$ (condição de degradação, com implicações de comportamento no barramento); e *Bus-Off*, quando $TEC \geq 256$, situação em que o nó é efetivamente desconectado do barramento [7, 18].

Como a análise desta métrica deve refletir o comportamento do sistema como um todo, os valores apresentados nesta seção consideram as quatro ECUs que efetivamente geram tráfego no barramento CAN durante o experimento. Para cada instante de observação (indexado pelo número acumulado de quadros transmitidos), define-se o pior caso instantâneo dos contadores como:

$$TEC_{\max}(n) = \max_{i \in \mathcal{E}} TEC_i(n) \quad \text{e} \quad REC_{\max}(n) = \max_{i \in \mathcal{E}} REC_i(n), \tag{5.4}$$

em que $\mathcal{E}$ é o conjunto das ECUs transmissoras avaliadas e n é o número acumulado de quadros transmitidos desde o início da execução.

A Figura 5.3 apresenta a evolução de $TEC_{\max}(n)$ e $REC_{\max}(n)$ ao longo do experimento, bem como os limiares de transição para *Error Passive* (128) e *Bus-Off* (256). Observa-se que o número máximo entre as ECUs permaneceu abaixo do limiar de *Error Passive* em todo o ensaio ($TEC_{\max}(n) < 128$ e

$REC_{\max}(n) < 128$), com picos aproximados de $TEC_{\max}(n) \lesssim 40$ e $REC_{\max}(n) \lesssim 20$, e não houve ocorrência de estado *Bus-Off*. Esses resultados sustentam, de forma conservadora, que a comunicação permaneceu em regime estável do ponto de vista de confinamento de falhas, sem degradação de estado em nenhuma das ECUs geradoras de tráfego no contexto experimental considerado.

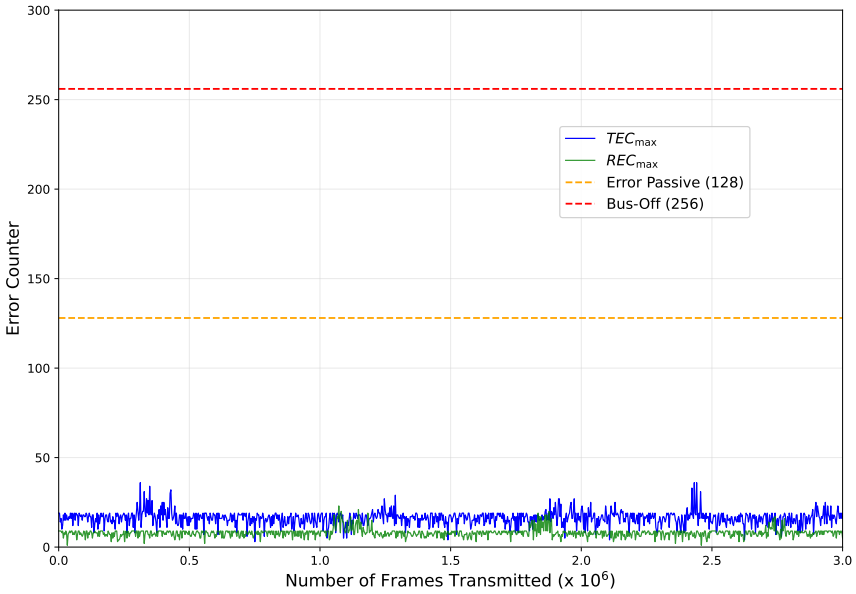

Figura 5.3: Máximo instantâneo dos contadores de erro entre as ECUs transmissoras: $TEC_{\max}(n)$ e $REC_{\max}(n)$, com indicação dos limiares de *Error Passive* (128) e *Bus-Off* (256). Fonte: elaborada pelo autor (2025).

5.2.4 Taxa de Perda de Arbitragem (Arbitration Loss Rate)

A taxa de perda de arbitragem (*arbitration loss rate*) foi monitorada para caracterizar a competição por acesso ao meio e o comportamento da priorização por identificador no barramento CAN. Diferentemente de redes do tipo Ethernet, o protocolo CAN realiza arbitragem bit a bit de forma não destrutiva: quando dois nós iniciam transmissão simultaneamente, o nó que transmite o identificador de menor valor (maior prioridade) mantém a transmissão, enquanto o nó de menor prioridade detecta a perda de arbitragem e adia sua tentativa, sem que o quadro vencedor seja corrompido [7, 18, 12, 6].

No CAN-ESP, a medição foi obtida a partir da instrumentação implementada na biblioteca `can_esp_lib`, contabilizando, para cada nó transmissor, o número de tentativas de transmissão ($N_{\text{tentativas}}$) e o número de quadros efetivamente transmitidos com sucesso ($N_{\text{tx\_ok}}$). A diferença $N_{\text{tentativas}} - N_{\text{tx\_ok}}$ representa eventos em que a transmissão não prosseguiu imediatamente devido à competição no acesso ao meio (perda de arbitragem e/ou adiamento de transmissão), sendo utilizada como estimador operacional para a taxa de perda de arbitragem no contexto experimental. A Figura 5.4 apresenta a taxa de perda de arbitragem cumulativa (acumulada desde o início da execução) ao longo do experimento, isto é, calculada a partir das contagens acumuladas de $N_{\text{tentativas}}$ e $N_{\text{tx\_ok}}$ até cada ponto do eixo x. O comportamento assintótico da curva demonstra a estabilidade estatística da rede sob alta carga.

$$\text{Arbitration Loss Rate}(\%) = \frac{N_{\text{tentativas}} - N_{\text{tx\_ok}}}{N_{\text{tentativas}}} \times 100 \tag{5.5}$$

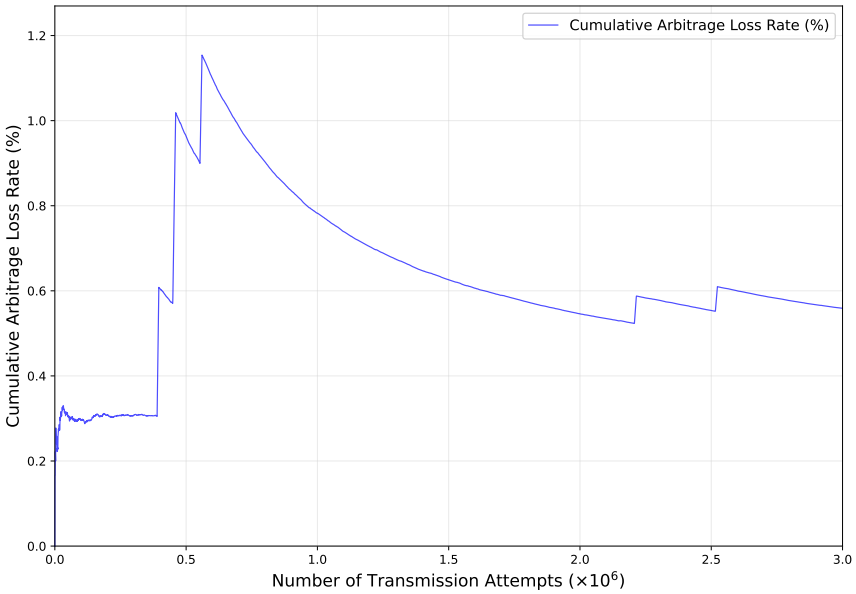

Figura 5.4: Taxa de Perda de Arbitragem cumulativa (Arbitration Loss Rate) em função do número de tentativas de transmissão. Fonte: elaborada pelo autor (2025).

Os resultados experimentais revelam que a taxa de perda de arbitragem cumulativa apresenta um comportamento assintótico, estabilizando-se em aproximadamente 0,55% ao atingir a marca de 3 000 000 tentativas de transmissão. Este valor reduzido é condizente com a carga moderada do barramento (10% a 25%), indicando que a probabilidade de disputa simultânea pelo acesso ao meio permanece controlada durante o experimento. A estabilidade da curva após as oscilações iniciais demonstra o determinismo da arquitetura CAN-ESP e a eficácia da biblioteca `can_esp_lib` no gerenciamento das filas de mensagens, garantindo que as disputas sejam resolvidas pelo controlador TWAI sem comprometer a integridade temporal do sistema. Esse comportamento valida a robustez do escalonamento por prioridades, assegurando que mensagens críticas obtenham acesso previsível ao barramento mesmo sob tráfego concorrente.

5.2.5 Latência da Comunicação (*Event Time Stamping*)

Sob as condições de carga definidas na subseção anterior, a latência média do sistema, mensurada entre a transmissão e a recepção das mensagens, variou de 129 μs a 141 μs, sendo o tempo teórico mínimo (ou latência teórica mínima) para transmissão de um quadro é de 128 μs. A variação de aproximadamente 9,3% encontra-se dentro dos padrões aceitáveis para redes CAN de alta velocidade (1 Mbps), conforme ilustra a Figura 5.5.

A latência foi calculada pela Equação 5.6, onde N_{bits} é o número total de bits do quadro (tamanho médio de 128 bits para o formato estendido de 29 bits, conforme estabelecido na Seção 5.1.4) e a taxa de transmissão de 1 Mbps. Os valores de latência medidos demonstram uma proximidade direta com o tempo teórico mínimo de 128 μs, e a variação observada é tipicamente atribuída ao *overhead* de *software* e processamento interno dos nós (por exemplo, enfileiramento de transmissão/recepção, atendimento de interrupções e manuseio de *buffers* no *driver*), bem como aos efeitos de arbitragem e ao bloqueio inerente ao CAN quando um quadro de menor prioridade já se encontra em transmissão [7, 18, 24, 53, 12]. Como a latência é obtida com referência temporal única por meio da instrumentação descrita na Seção 5.1.2, os

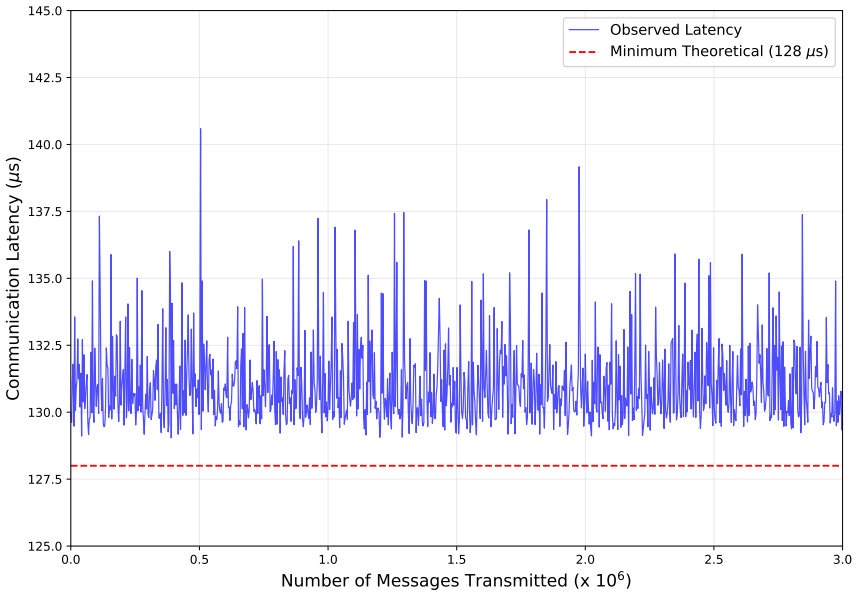

Figura 5.5: Latência Média do Sistema (*Event Time Stamping*). Fonte: elaborada pelo autor (2025).

resultados não dependem de sincronização de relógios entre ECUs.

$$T_{\text{frame}} = \frac{N_{\text{bits}}}{\text{Taxa de transmissão (bps)}} \tag{5.6}$$

Os valores observados são compatíveis com a execução de comandos críticos, tais como os de frenagem e aceleração, no contexto experimental avaliado.

5.2.6 Tempo de Resposta do Sistema (*System Response Time*)

O tempo de resposta do sistema é um parâmetro crítico para avaliar o desempenho de uma rede CAN em aplicações automotivas. Ele pode ser definido como a soma do tempo necessário para a transmissão do quadro no barramento, do tempo de propagação do sinal e do tempo de processamento da ECU receptora antes de gerar uma resposta. O tempo de resposta do sistema foi analisado para verificar a capacidade da rede CAN-ESP em lidar com variações na carga operacional. A análise demonstrou que a comunicação entre ECUs ocorreu de maneira síncrona e eficiente, com variação do tempo de resposta observado dentro da faixa apresentada na Figura 5.6, mesmo sob carga elevada.

O tempo de resposta do sistema foi modelado conforme a Equação 5.7, na qual T_{frame} representa o tempo de transmissão do quadro, $T_{\text{processing}}$ corresponde ao tempo de processamento na ECU receptora e $T_{\text{propagation}}$ denota o tempo de propagação do sinal no barramento.

$$T_{\text{response}} = T_{\text{frame}} + T_{\text{processing}} + T_{\text{propagation}} \tag{5.7}$$

O tempo de transmissão de um quadro CAN é determinado pelo seu tamanho e pela taxa de transmissão do barramento, conforme a equação:

$$T_{\text{transmissão}} = \frac{\text{Tamanho do quadro (bits)}}{\text{Taxa de transmissão (bps)}}. \tag{5.8}$$

No sistema CAN-ESP, cada quadro utiliza o formato *Extended Frame* (ID de 29 bits) com um *payload* de 8 bytes. O tamanho total do quadro de dados varia de um mínimo de 64 bits (para 0 bytes de dados) a um máximo de 131 bits (para 8 bytes de dados), excluindo bits de *stuffing*, conforme explicado na Seção 2.3. Durante o experimento, o tamanho médio dos quadros transmitidos foi de 128 bits e a taxa de transmissão adotada, conforme explicitado anteriormente, é de 1 Mbps, resultando em:

$$T_{\text{transmissão}} = \frac{128}{1.000.000} = 128\,\mu\text{s}. \tag{5.9}$$

O tempo de propagação do sinal no barramento depende do comprimento da rede e da velocidade de propagação da onda eletromagnética no cabo, sendo expresso por:

$$T_{\text{propagação}} = \frac{\text{Comprimento do barramento (m)}}{\text{Velocidade de propagação (m/s)}}. \tag{5.10}$$

Para um barramento de 10 metros, utilizando um cabo UTP CAT-5 com velocidade de propagação típica de aproximadamente 5 ns/m, obtém-se:

$$T_{\text{propagação}} = 10 \times 5\,\text{ns} = 50\,\text{ns} = 0.05\,\mu\text{s}. \tag{5.11}$$

O tempo de processamento da ECU varia conforme a arquitetura do microcontrolador. No caso do ESP32 WROOM-32, considerando um clock da CPU de 240 MHz, um tempo médio por instrução de aproximadamente 4 ns e cerca de 5.000 ciclos para processar uma mensagem CAN e gerar uma resposta, tem-se:

$$T_{\text{processamento}} = 5.000 \times 4\,\text{ns} = 20\,\mu\text{s}. \tag{5.12}$$

Somando os tempos calculados, o tempo total de resposta é:

$$T_{\text{resposta}} = 128\,\mu\text{s} + 0.05\,\mu\text{s} + 20\,\mu\text{s} \approx 148\,\mu\text{s}. \tag{5.13}$$

O tempo de resposta obtido para o CAN-ESP encontra-se na ordem de grandeza compatível com requisitos temporais típicos de funções de controle veicular, os quais são comumente definidos em janelas de milissegundos (por exemplo, ciclos de controle na faixa de 10 a 50 ms). Assim, os atrasos observados no nível de centenas de microssegundos indicam ampla margem temporal para a execução dessas funções no contexto experimental considerado, respeitando os mecanismos de arbitragem e as limitações de não-preempção inerentes ao CAN [7, 18, 24, 53, 12]. Em consonância com a latência medida na Seção 5.2.5, observa-se variação temporal limitada na comunicação, reforçando a consistência entre a modelagem simplificada de tempo de resposta e o comportamento observado no sistema experimental. Dessa forma, no contexto avaliado, o CAN-ESP apresenta margem temporal compatível com requisitos típicos de controle

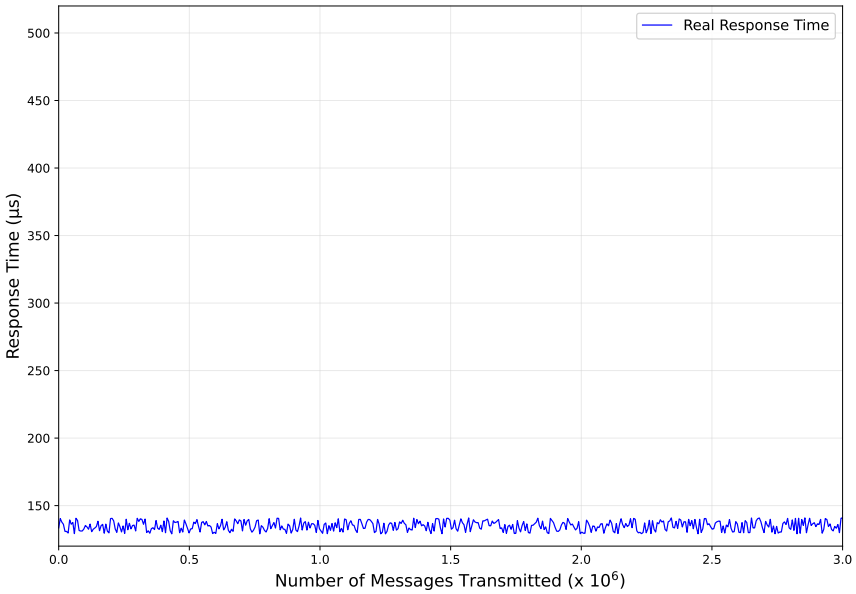

Figura 5.6: Tempo de Resposta do Sistema (*System Response Time*) (observado). Fonte: elaborada pelo autor (2025).

veicular em escala de milissegundos, respeitando os efeitos de arbitragem e o caráter não preemptivo do CAN [7, 18, 24, 53, 12].

Na sequência, as Seções 5.2.7 e 5.2.8 caracterizam a variabilidade estatística da latência observada, complementando a análise temporal com uma visão de dispersão e do pior caso observado sob a condição considerada.

5.2.7 Análise Estatística do *Jitter* (*Jitter Analysis*)

Para avaliar a estabilidade temporal e a previsibilidade da arquitetura CAN-ESP, foi conduzida uma análise estatística da variabilidade da latência de comunicação. Neste trabalho, adota-se como métrica de variação temporal o *excesso de latência* em relação a um limite inferior de referência, definido por:

$$J = T_{\text{observado}} - T_{\min}, \tag{5.14}$$

em que $T_{\text{observado}}$ é a latência medida para cada mensagem e $T_{\min}$ representa o tempo mínimo teórico de transmissão do quadro no barramento (aqui utilizado como referência, $T_{\min} = 128\ \mu s$). Essa definição quantifica, de forma direta, o atraso adicional introduzido por arbitragem, bloqueio e *overhead* de *software* sob tráfego concorrente, sendo adequada para caracterizar a previsibilidade temporal em redes CAN [7, 18, 24, 53, 12]. A Figura 5.7 apresenta o histograma da densidade de probabilidade da variável J para uma amostra de $3\,000\,000$ mensagens transmitidas sob o cenário de referência S1 (*baseline* – 20% a 80%). Na figura, a linha tracejada vermelha indica a média ($\mu = 1.93 \mu s$), enquanto a área sombreada em cinza representa o intervalo $\mu \pm \sigma$. Observa-se um comportamento determinístico, com uma cauda longa decorrente da arbitragem e do *bit stuffing* do barramento.

A análise quantitativa revela uma média de J de $\mu = 1.93\ \mu s$ com um desvio padrão de $\sigma = 0.65\ \mu s$. A distribuição apresenta uma assimetria positiva, com uma cauda à direita característica. É fundamental

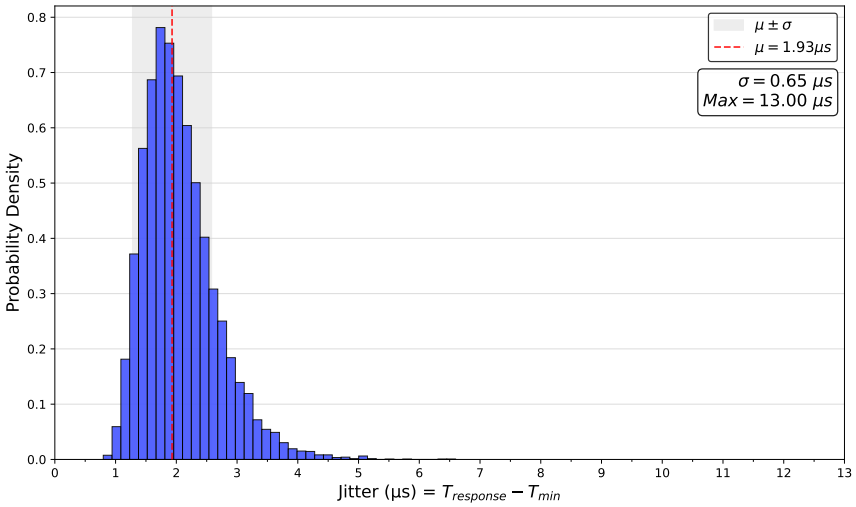

Figura 5.7: Distribuição da densidade de probabilidade do *jitter* de comunicação observado. Fonte: elaborada pelo autor (2025).

notar que, conforme discutido na Seção 2.3, essa variabilidade não decorre exclusivamente da competição pelo barramento (arbitragem), mas é intrínseca ao mecanismo de *bit stuffing* do protocolo CAN. Como o conteúdo dos dados varia aleatoriamente, a inserção dinâmica de bits de sincronização altera o tempo de transmissão de cada quadro, contribuindo para o *jitter* observado mesmo sob tráfego operacional elevado.

No pior caso registrado, o excesso de latência atingiu $J_{\max} = 13.00\ \mu s$, resultando em uma latência total de $141\ \mu s$. Este valor está dentro do limite teórico máximo de variação do quadro (que pode chegar a 160 bits ou $160\ \mu s$ no pior caso de *stuffing*), sugerindo que, na carga de 20% a 80% avaliada, o sistema CAN-ESP operou majoritariamente livre de bloqueios severos de arbitragem. O resultado permanece muito inferior às janelas temporais de controle veicular (da ordem de dezenas de milissegundos), validando a robustez temporal da solução no contexto experimental avaliado.

5.2.8 Análise do Pior Caso de Tempo de Resposta (WCRT)

Enquanto a análise de *jitter* caracteriza a estabilidade estatística da comunicação, a avaliação de segurança e tempo real requer verificar se mensagens críticas permanecem dentro de limites temporais (*deadlines*) sob condições de alta ocupação do barramento. Essa verificação foi realizada por meio da análise do pior caso observado de tempo de resposta no cenário de estresse máximo (S4), complementada pela discussão dos fatores de bloqueio e interferência previstos para redes CAN priorizadas por identificador [24, 53, 12]. Esta métrica é definida como o Pior Caso de Tempo de Resposta (*Worst-Case Response Time* – WCRT) que, durante os experimentos, variaram de $141\ \mu s$ (mensagem de maior prioridade, 0x001) a $680\ \mu s$ (mensagem de menor prioridade, 0x603), confirmando a efetividade do mecanismo de arbitragem por identificador.

Para validar a eficácia do mecanismo de arbitragem, mediu-se o tempo máximo de resposta sob o cenário de estresse máximo. Nesta configuração, a rede CAN foi submetida a uma condição de alta ocupação (*bus load* $> 90\%$) gerada através da injeção de tráfego sintético de fundo e intensificação controlada do envio de mensagens de baixa prioridade. Dessa forma, a transmissão de dados de frenagem disputou o

acesso ao meio contra tráfego CAN concorrente, em um cenário deliberadamente adverso para caracterização do pior caso observado. A Figura 5.8 apresenta a comparação entre o tempo médio e o WCRT observado para todas as mensagens do sistema. Ressalta-se que, nos ensaios deste capítulo, as mensagens CAN foram transmitidas no formato estendido (identificador de 29 bits), conforme definido na especificação CAN 2.0 e na norma ISO 11898. Por convenção de apresentação, os identificadores são exibidos em hexadecimal sem zeros à esquerda; assim, por exemplo, `0x001` corresponde ao identificador estendido `0x00000001` [7, 18].

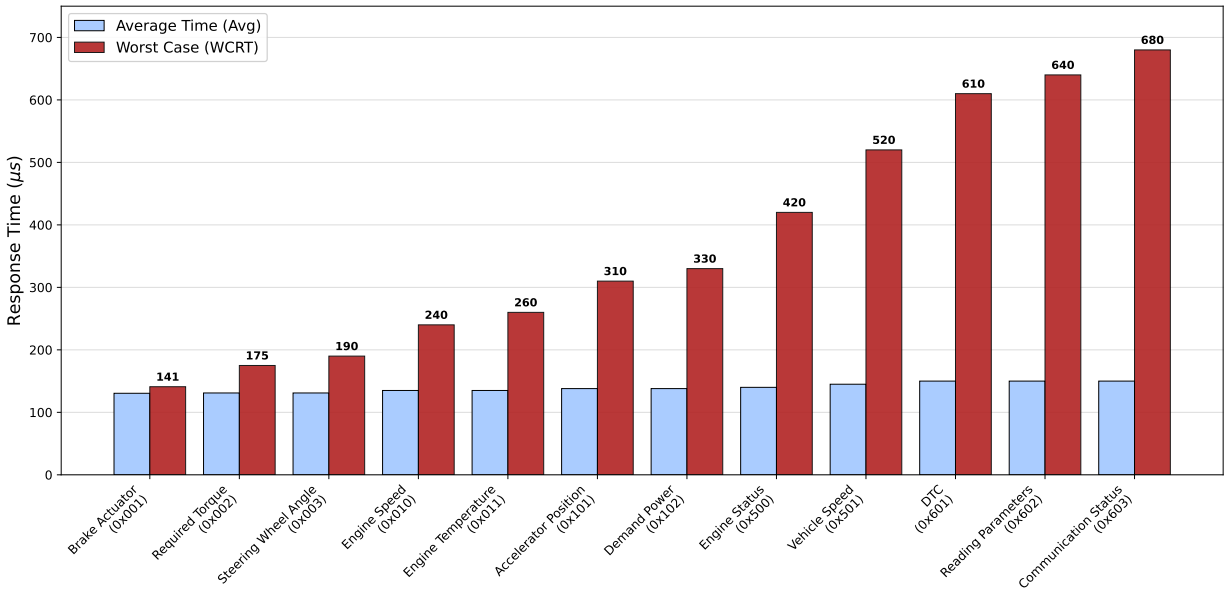

Figura 5.8: Comparação entre Tempo Médio e Pior Caso de Tempo de Resposta (WCRT) para as mensagens da rede CAN-ESP. Fonte: elaborada pelo autor (2025).

Os resultados indicam elevada previsibilidade temporal para as funções de segurança no cenário avaliado. O pior caso observado de tempo de resposta (WCRT) da mensagem de Pressão de Freio (ID estendido `0x00000001`, apresentado como `0x001`) foi de 141 μs, valor consistente com a latência máxima observada nos testes descritos na Seção 5.2.5. Adicionalmente, nota-se no gráfico que, para as mensagens de alta prioridade (IDs `0x001` a `0x003`), a diferença entre o tempo médio e o WCRT é mínima, demonstrando um comportamento determinístico onde o *jitter* é praticamente desprezível para as funções críticas. Esse comportamento é compatível com a priorização por identificador no CAN, que favorece mensagens críticas na arbitragem; entretanto, por se tratar de um barramento não preemptivo, tal previsibilidade deve ser interpretada à luz do bloqueio e da interferência possíveis no pior caso teórico [7, 18, 24, 53, 12].

Em contraste, mensagens de menor prioridade, como as de Diagnóstico (IDs estendidos `0x0000060X`, apresentados como `0x60X`), sofreram atrasos significativos devido à arbitragem, atingindo picos de até 680 μs. Esse comportamento evidencia que a arquitetura CAN-ESP preserva a hierarquia de prioridades do CAN, favorecendo mensagens críticas na arbitragem e deslocando o impacto do tráfego concorrente principalmente para mensagens de menor prioridade. Nos experimentos realizados, comandos de atuação crítica (Freio e Torque) permaneceram com atrasos baixos (tipicamente < 200 μs) mesmo sob alta ocupação do barramento, enquanto fluxos de diagnóstico apresentaram maior degradação temporal [18, 24, 53].

5.2.9 Ameaças à Validade, Limitações e Considerações sobre Segurança

Embora os resultados experimentais indiquem comportamento determinístico e robustez lógica no domínio temporal, é importante discutir as limitações do *hardware Commercial Off-The-Shelf* (COTS) utilizado face aos requisitos de segurança funcional automotiva. O SoC ESP32 e o transceptor SN65HVD230 não possuem certificação ISO 26262 (ASIL – *Automotive Safety Integrity Level*), o que os torna inadequados para sistemas críticos (e.g., freio e direção, tipicamente ASIL C/D) em veículos comerciais de passageiros. Nesses contextos, a redundância de *hardware*, a imunidade a falhas transientes (*Soft Errors*) e processos de desenvolvimento qualificados são mandatórios.

A utilização de um transceptor de 3.3V (modelo SN65HVD230), embora plenamente compatível com a norma ISO 11898-2, apresenta, em tese, menor imunidade a interferência eletromagnética (EMI) severa em comparação com transceptores legados de 5V, devido à menor margem de ruído. Consequentemente, a arquitetura CAN-ESP, em sua implementação física atual, tem aplicabilidade comprovada especificamente para domínios de baixa criticidade e prototipagem, tais como Veículos Elétricos Leves (LEVs) e ambientes de desenvolvimento de conceitos SDV. Para uma eventual transição deste modelo arquitetural para veículos de estrada comerciais, recomenda-se a substituição dos nós críticos por microcontroladores e transceptores de grau automotivo (e.g., famílias Infineon Aurix ou NXP S32K), mantendo-se, contudo, a lógica de orquestração OTA e a topologia distribuída, contribuições centrais deste trabalho.

Adicionalmente, é crucial distinguir a segurança da informação (*security*) da segurança operacional (*safety*) no contexto da arquitetura proposta. Embora o protocolo OTA nativo utilize HTTPS/TLS para garantir a confidencialidade e integridade do *firmware* durante o transporte via Wi-Fi, a arquitetura interna do barramento CAN opera sob um modelo de confiança implícita entre os nós, uma característica intrínseca ao padrão CAN 2.0B. Consequentemente, o comprometimento físico ou lógico de uma única ECU poderia resultar não apenas em ataques de negação de serviço (DoS) do tipo *Babbling Idiot*, mas também em ataques de *Spoofing* (falsificação de identidade) e *Replay*. Vale destacar que a atual implementação priorizou o desempenho em tempo real e a compatibilidade estrita com a norma ISO 11898, uma vez que a introdução de criptografia ou códigos de autenticação de mensagem (MACs) nos *frames* padrão de 8 bytes imporia um *overhead* proibitivo para a latência, exigindo estratégias complexas de fragmentação ou a migração para CAN-FD (*Flexible Data-Rate*). Reconhece-se, portanto, que a robustez contra ataques cibernéticos intencionais constitui uma limitação conhecida da presente implementação, cujo foco reside em estabelecer a viabilidade da orquestração OTA e do modelo distribuído sob restrições temporais rígidas [54].

Em síntese, as limitações discutidas — relacionadas à necessidade de validação do *hardware*, à imunidade eletromagnética e aos mecanismos de segurança cibernética — delimitam com precisão o escopo de aplicabilidade imediata da arquitetura CAN-ESP: um protótipo de referência para sistemas de baixa criticidade e pesquisa em arquiteturas SDV. Todas as análises de desempenho e conclusões deste trabalho devem ser interpretadas dentro desse contexto. As contribuições principais, contudo, são intencionalmente dissociadas dessas limitações de implementação, podendo ser transpostas para plataformas de grau automotivo em trabalhos futuros.

5.3 DISCUSSÃO

Os resultados indicam que a arquitetura CAN-ESP mantém comportamento previsível mesmo em condições adversas de alta ocupação do barramento (*bus load* $> 90\%$, Seção 5.2.1), na qual a competição por arbitragem é intensificada por tráfego CAN concorrente. Nesse cenário, a hierarquia de prioridades por identificador se refletiu na separação clara entre fluxos críticos e não críticos: enquanto mensagens de atuação permaneceram com pior caso observado de tempo de resposta baixo (WCRT observado de $141\ \mu s$ para o freio, Seção 5.2.8), fluxos de menor prioridade exibiram maior degradação temporal. A interpretação, contudo, deve considerar a não-preempção do CAN e os fatores de bloqueio e interferência previstos no pior caso teórico, conforme discutido na literatura clássica [24, 53, 12].

Adicionalmente, a estabilidade temporal foi evidenciada pelo baixo *jitter* médio de $1.93\ \mu$s (Seção 5.2.7), enquanto a integridade da camada física foi atestada pela ausência de estados de *bus-off* (Seção 5.2.3) e por uma taxa média global de retransmissão da ordem de 15 ppm ($0{,}0015\%$) (Seção 5.2.2), qualificando a solução para aplicações de controle em tempo real.

A arquitetura proposta representa uma plataforma viável e robusta para o avanço do paradigma do Veículo Definido por *Software* (SDV) fora dos limites da P&D industrial em grande escala. No quesito escalabilidade (Seção 5.1.3), os resultados indicam que a solução mantém as propriedades temporais observadas quando submetida a condições de maior concorrência e ocupação do barramento, sugerindo viabilidade de expansão do número de nós e fluxos de comunicação, desde que preservadas as restrições de camada física e mantida a carga do barramento em patamares compatíveis com os requisitos temporais do sistema. Contudo, é fundamental ressaltar que, embora a arquitetura seja logicamente escalável, a expansão física do número de nós está condicionada a fatores como topologia, comprimento do barramento, derivações (*stubs*) e orçamento de carga, além das características elétricas dos transceptores COTS utilizados (Seção 5.2.9). Sendo assim, a determinação desses limites físicos constitui uma extensão natural deste trabalho [18, 12, 6].

6 CONCLUSÃO

Este trabalho demonstrou a viabilidade da CAN-ESP, uma arquitetura para redes intraveiculares que responde ao desafio de prover comunicação com comportamento temporal previsível e capacidades de gestão de *software* (OTA) em uma plataforma de baixo custo e código aberto. Ao integrar o barramento CAN conforme a norma ISO 11898 com as capacidades de processamento e conectividade do SoC ESP32, o projeto atende ao objetivo de reduzir as barreiras de custo e reprodutibilidade sem sacrificar os requisitos funcionais essenciais de uma rede de tempo real. Para além do *hardware* de baixo custo e do caráter *open source*, a CAN-ESP implementa um padrão arquitetural replicável para orquestração de atualizações OTA distribuídas e resilientes, dissociando, em nível arquitetural, a gestão do ciclo de vida do *software* das funções de comunicação de controle em tempo real, de modo a minimizar impactos sobre as mensagens críticas, o que favorece a adoção em contextos com restrição de orçamento e infraestrutura.

No que concerne aos objetivos específicos de validação de desempenho e robustez, os experimentos demonstraram a viabilidade técnica da arquitetura sob condições de carga e cenários de estresse elevados. A análise de desempenho evidenciou o comportamento esperado do mecanismo de arbitragem por prioridade: a ocorrência de perda de arbitragem permaneceu consistente com o processo normal de contenção do CAN (Seção 5.2.4), sem indícios de anomalias que comprometessem o atendimento temporal do tráfego de mensagens no barramento. As mensagens críticas apresentaram um WCRT máximo observado de 141 μs (Seção 5.2.8), em concordância com o modelo teórico de tempo de resposta do sistema (Seção 5.2.6) e apresentando comportamento estável mesmo sob saturação. Adicionalmente, a estabilidade temporal foi evidenciada pelo baixo *jitter* médio de 1.93 μs (Seção 5.2.7), enquanto a integridade da camada física foi atestada pela ausência de estados de *bus-off* (Seção 5.2.3) e por uma taxa de retransmissão residual inferior a 0.0015% (Seção 5.2.2), indicando potencial para aplicações de controle em tempo real no escopo e nas condições experimentais consideradas.

Em síntese, os resultados obtidos sustentam a viabilidade de desenvolver e validar uma arquitetura CAN de baixo custo que preserve requisitos de tempo real e, simultaneamente, integre funcionalidades alinhadas ao paradigma do Veículo Definido por *Software*, como atualização OTA. Além de responder à questão de pesquisa e cumprir os objetivos delineados, este trabalho consolida contribuições reprodutíveis (*hardware* homogêneo e a biblioteca `can_esp_lib` de código aberto) e um conjunto de métricas quantitativas que fortalecem a evidência experimental. Como limitação inerente ao escopo, a validação foi conduzida no contexto de um protótipo e de uma plataforma veicular específica, o que motiva a ampliação de testes para topologias, densidades e condições ambientais mais diversas.

Ao tornar acessível uma plataforma de comunicação intraveicular com comportamento temporal previsível e capacidades de gestão de *software*, este trabalho contribui para a democratização do desenvolvimento de sistemas veiculares, contribuindo para o avanço da inovação e da pesquisa em Veículos Definidos por *Software* (SDV) em contextos de pesquisa acadêmica e projetos com orçamento limitado.

6.1 TRABALHOS FUTUROS

Destacamos que a análise de escalabilidade realizada para o subsistema de atualização remota indica que o protocolo de distribuição OTA em rede *mesh* pode atender a cenários com dezenas de ECUs (e.g., 50 nós) sem comprometer janelas operacionais típicas de manutenção. Para densidades superiores, especialmente em arquiteturas zonais de próxima geração, sugere-se que investigações futuras explorem a segmentação em múltiplas sub-malhas (*clusters*) operando em canais Wi-Fi ortogonais, bem como estratégias de coordenação entre domínios. Adicionalmente, recomenda-se a validação da solução em cenários de alta interferência eletromagnética, com chicotes maiores e condições ambientais mais severas.

Complementarmente, configuram-se como oportunidades de pesquisa a ampliação dos mecanismos de segurança já discutidos ao longo do trabalho. Recomenda-se o desenvolvimento de Sistemas de Detecção de Intrusão (IDS) baseados em anomalias de tráfego e a investigação de mecanismos leves de autenticação de mensagens no barramento (e.g., MACs truncados, contadores de frescor), buscando equilíbrio entre *overhead* e garantias de integridade/autenticidade. Adicionalmente, a integração com protocolos V2X (*Vehicle-to-Everything*) pode habilitar a troca de informações coordenadas com outros veículos (V2V), infraestrutura viária (V2I), pedestres (V2P) e redes (V2N).

REFERÊNCIAS BIBLIOGRÁFICAS

1 TINDELL, K. *CAN bus frame with stuff bit and correct CRC*. 2021. Canis Automotive Labs Ltd. Licensed under CC BY-SA 4.0. Disponível em: <https://commons.wikimedia.org/wiki/File:CAN-bus-frame-with-stuff-bit-and-correct-CRC.png>. Acesso em: 9 jul. 2025.

2 CONTRIBUTORS, W. *CAN Bus frame in base format without stuff bits*. 2024. Disponível em: <https://en.wikipedia.org/wiki/File:CAN-Bus-frame_in_base_format_without_stuffbits.svg>. Acesso em: 5 fev. 2025.

3 WOO, S.; MOON, D.; YOUN, T.-Y.; LEE, Y.; KIM, Y. Can id shuffling technique (cist): Moving target defense strategy for protecting in-vehicle can. *IEEE Access*, v. 7, p. 15521–15536, 2019. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8631032>. Acesso em: 5 jul. 2025.

4 CORRIGAN, S. *Introduction to the Controller Area Network (CAN)*. [S.l.], 2008. Application Report. Disponível em: <https://www.ti.com/lit/pdf/slla270>. Acesso em: 9 jul. 2025.

5 BROY, M. Challenges in automotive software engineering. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 2*. [S.l.]: IEEE Press, 2015. (ICSE '15), p. 33—42.

6 LAWRENZ, W. (Ed.). *CAN System Engineering: From Theory to Practical Applications*. London: Springer London, 2013.

7 Robert Bosch GmbH. *CAN Specification 2.0*. Stuttgart: Robert Bosch GmbH, 1991. Part A and B. Disponível em: <http://esd.cs.ucr.edu/webres/can20.pdf>. Acesso em: 5 nov. 2023.

8 ZUBERI, K.; SHIN, K. Real-time decentralized control with can. In: *Proceedings 1996 IEEE Conference on Emerging Technologies and Factory Automation. ETFA '96*. [S.l.: s.n.], 1996. v. 1, p. 93–99.

9 TINETTI, F. G.; ROMERO, F. L.; PÉREZ, A. D. Can bus experiments of real-time communications. In: SPRINGER. *Computer Science–CACIC 2017: 23rd Argentine Congress, La Plata, Argentina, October 9-13, 2017, Revised Selected Papers 23*. [S.l.], 2018. p. 253–262.

10 PENG, H. Electric vehicle control system based on can bus. *Acta Technica*, Prague, v. 1A, p. 541–552, 2017. Disponível em: <http://journal.it.cas.cz/62(2017)-1A/Paper%2050%20Han%20Peng.pdf>. Acesso em: 5 nov. 2023.

11 GOSWAMI, P. *The Software-defined Vehicle and Its Engineering Evolution: Balancing Issues and Challenges in a New Paradigm of Product Development*. [S.l.], 2024.

12 NATALE, M. D.; ZENG, H.; GIUSTO, P.; GHOSAL, A. *Understanding and Using the Controller Area Network Communication Protocol*. Springer New York, 2012. Disponível em: <https://link.springer.com/book/10.1007%2F978-1-4614-0314-2>. Acesso em: 5 nov. 2023.

13 DESAI, M. et al. Controller area network for intelligent vehicular systems. In: IEEE. *2013 International Conference on Advances in Technology and Engineering (ICATE)*. [S.l.], 2013. p. 1–6.

14 SASE, A. A.; BHATESHVAR, Y. K.; VORA, K. C. Electric vehicle control system by using controller area network communication. *International Journal of Engineering Sciences*, v. 15, n. 2, p. 81–88, 2022.

15 COELHO, A.; SILVA-FILHO, A. G. Savips: Uma infraestrutura veicular autônoma em pequena escala para adas. *Results in Engineering*, v. 17, p. 100969, 2023. Publicado online em 2023, volume de 2024.

16 MEAH, K.; II, D. H.; SMITH, A.; HOCK, P.; WALSH, A. Design and implementation of the software and hardware system for a formula sae electric vehicle. In: *2020 IEEE International Conference on Services Computing (SCC)*. [S.l.: s.n.], 2020. p. 132–137.

17 PEREIRA, D. M.; RODRIGUES-FILHO, R.; GONçALVES, V.; SANTOS, H. dos; TRINDADE, R.; MENEGUETTE, R.; SERRANO, A. L.; FILHO, G. R. Can-esp: Rede can de baixo custo para veículos elétricos. In: *Anais do IX Workshop de Computação Urbana*. Porto Alegre, RS, Brasil: SBC, 2025. p. 1–14. ISSN 2595-2706. Disponível em: <https://sol.sbc.org.br/index.php/courb/article/view/35247>.

18 ISO. *ISO 11898:2015: Road vehicles – Controller area network (CAN)*. 2015. International Organization for Standardization, Geneva, Switzerland. Disponível em: <https://www.iso.org/standard/63648.html>. Acesso em: 5 fev. 2025.

19 HPL, S. C. Introduction to the controller area network (can). *Application Report SLOA101*, Texas instruments Dallas, TX, USA, p. 1–17, 2002.

20 CAN in Automation (CiA) e.V. *CiA 301: CANopen Application Layer and Communication Profile*. [S.l.], 2002. Disponível em: <https://workarea.ego-gw.it/ego2/ego/itf/software/301_canopen.pdf>. Acesso em: 9 jul. 2025.

21 ZIMMERMANN, W.; SCHMIDGALL, R. *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur*. 5. ed. Wiesbaden: Springer Vieweg, 2014.

22 BURNS, A.; WELLINGS, A. *Real-Time Systems and Programming Languages*. 4. ed. Harlow, England: Addison-Wesley, 2009. ISBN 978-0321417459.

23 BUTTAZZO, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. 3. ed. New York: Springer, 2011. (Real-Time Systems Series). ISBN 978-1461406754.

24 TINDELL, K.; BURNS, A.; WELLINGS, A. Calculating controller area network (can) message response times. *Control Engineering Practice*, v. 3, n. 8, p. 1163–1169, 1995. ISSN 0967-0661. Disponível em: <https://www.sciencedirect.com/science/article/pii/0967066195001128>.

25 Espressif Systems. *Two-Wire Automotive Interface (TWAI)*. 2025. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/twai.html#signal-lines-and-transceiver>. Acesso em: 5 fev. 2025.

26 CHIKHALE, S. N. Automobile design and implementation of can bus protocol - a review. *IJRDO-Journal of Electrical And Electronics Engineering*, v. 4, n. 1, January 2018. ISSN 2456-6055.

27 SHARMA, P. K.; SINGH, S.; JEONG, Y.-S.; PARK, J. H. A lightweight multi-layered security framework for IoT networks using a lightweight cryptographic algorithm. *The Journal of Supercomputing*, Springer, v. 76, n. 8, p. 5837–5859, 2020.

28 Espressif Systems. *ESP-WIFI-MESH*. 2025. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html>. Acesso em: 10 fev. 2025.

29 LI, R.; WU, J.; WANG, H. Design method of can bus network communication structure for electric vehicle. In: *IFOST 2010 Proceedings*. [S.l.: s.n.], 2010. p. 4.

30 VIJAYA, D.; KOKANE, R.; KALYANKAR, S. B. Implementation of the can bus in the vehicle based on arm 7. *International Journal of Research in Engineering and Technology*, v. 4, n. 1, p. 29–31, jan 2015. Disponível em: <http://www.ijret.org>.

31 WAGH, P.; PAWAR, R.; NALBALWAR, S. Vehicle speed control and safety prototype using controller area network. In: *2017 International Conference on Computational Intelligence in Data Science (ICCIDS)*. [S.l.: s.n.], 2017. p. 1–5.

32 ALZAHRANI, A.; WANGIKAR, S. M.; INDRAGANDHI, V.; SINGH, R. R.; SUBRAMANIYASWAMY, V. Design and implementation of sae j1939 and modbus communication protocols for electric vehicle. *Machines*, v. 11, n. 201, 2023. Disponível em: <https://doi.org/10.3390/machines11020201>.

33 ISMAIL, K.; MUHARAM, A.; AMIN; KALEG, S. Design of can bus for research applications purpose hybrid electric vehicle using arm microcontroller. *Energy Procedia*, v. 68, p. 288–296, 2015.

34 WANG, Y. Design of electric drive system of electric vehicle based on can bus. *Journal of Physics: Conference Series*, v. 1982, p. 012131, 2021.

35 JIANG, M. Design of electric vehicle control system based on can bus. In: *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. [S.l.: s.n.], 2022. p. 1288–1291.

36 SALUNKHE, A. A.; KAMBLE, P. P.; JADHAV, R. Design and implementation of can bus protocol for monitoring vehicle parameters. In: *IEEE International Conference On Recent Trends In Electronics Information Communication Technology*. India: [s.n.], 2016. p. 1–5.

37 KUMAR, H. S.; HEGDE, S. G.; ADIGA, S.; HUGAR, D. S.; S, G. Autonomous car using raspberry pi and can bus protocol. *International Journal of Innovative Science and Research Technology*, v. 8, n. 5, p. 3114–3119, 2023. ISSN 2456-2165.

38 MOHAMMED, M. S.; ABDULJABAR, A. M.; FAISAL, M. M.; MAHMMOD, B. M.; ABDULHUSSAIN, S. H.; KHAN, W.; LIATSIS, P.; HUSSAIN, A. Low-cost autonomous car level 2: Design and implementation for conventional vehicles. *Results in Engineering*, Elsevier, v. 17, p. 100969, 2023. Disponível em: <https://doi.org/10.1016/j.rineng.2023.100969>.

39 PHAM, N. N.; LEUCHTER, J.; PHAM, K. L.; DONG, Q. H. Battery management system for unmanned electric vehicles with can bus and internet of things. In: *2022 13th International Conference on Electrical Engineering (ICEENG)*. [S.l.: s.n.], 2022. v. 4, n. 3, p. 639–662.

40 NASR, A.; GHONEIMA, M.; ABDULLAH, B. A. Automotive software self reprogramming ota. In: *2022 13th International Conference on Electrical Engineering (ICEENG)*. [S.l.: s.n.], 2022. p. 76–80.

41 GIRON, J. D.; SERMENO, B. S.; SANTIAGO, A. T.; YAGO, J. A. N.; DOMINGO, M. A. B.; TAYO, L. A. S.; TRIA, L. A. R. Development of a mobile application-based system diagnostics and monitoring for a battery electric vehicle. In: *2023 IEEE Transportation Electrification Conference and Expo, Asia-Pacific (ITEC Asia-Pacific)*. [S.l.: s.n.], 2023.

42 SHAMSHIRI, R. R.; NAVAS, E.; DWORAK, V.; CHEEIN, F. A. A.; WELTZIEN, C. A modular sensing system with canbus communication for assisted navigation of an agricultural mobile robot. *Computers and Electronics in Agriculture*, v. 223, p. 109112, 2024.

43 AMMAR, M.; DJAMEL, R.; FATEH, M.; DJEMAI, K. Analyzing real-time communication: Arduino-based controller area network (can) in electric vehicle. *Studies in Engineering and Exact Sciences*, v. 5, n. 2, p. 01–33, 2024.

44 AKYILDIZ, I. F.; WANG, X.; WANG, W. A Survey on Wireless Mesh Networks. *IEEE Communications Magazine*, v. 43, n. 9, p. S23–S30, Sept. 2005.

45 SILVA, L. D. C. B.; NETO, G. R. d. S. F.; SILVA, R. D. da S. e; FREITAS, E. P. de; CARNEIRO, G. D. D. C. e. A flexible and scalable firmware update management system for IoT devices. *IEEE Latin America Transactions*, IEEE, v. 17, n. 09, p. 1533–1541, 2019.

46 AL-KASHOASH, H. A. MQTT-SN: A promising protocol for resource-constrained IoT systems. In: IEEE. *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. [S.l.], 2016. p. 646–651.

47 SHAH, S. S. H.; BASHIR, A. K.; AL-OTAIBI, M. A.; AL-GHAMDI, A. A. S. MQTT-based reliable data dissemination for vehicular ad-hoc networks. *IEEE Access*, IEEE, v. 8, p. 99341–99351, 2020.

48 KARUNARATNE, T.; KALUPAHANA, C.; DAS, S. Securing automotive over-the-air updates: A survey. In: IEEE. *2022 4th International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*. [S.l.], 2022. p. 1–6.

49 KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach*. 7th. ed. Boston: Pearson, 2016.

50 OLIVEIRA, C. G. M. B. de; ROCHA, A. F. da; MORAES, L. F. M. de. A survey of over-the-air update solutions for the internet of things. *ACM Computing Surveys*, v. 54, n. 6, 2021.

51 United Nations Economic Commission for Europe (UNECE). *UN Regulation No. 155: Uniform provisions concerning the approval of vehicles with regard to cyber security and cyber security management system*. 2021. Date of entry into force: 22 January 2021. Disponível em: <https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security>. Acesso em: 5 fev. 2025.

52 GUPTA, P.; KUMAR, P. The capacity of wireless networks. *IEEE Transactions on Information Theory*, v. 46, n. 2, p. 388–404, 2000.

53 DAVIS, R. I.; BURNS, A.; BRIL, R. J.; LUKKIEN, J. J. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, Springer, v. 35, p. 239–272, 2007.

54 AL-HAMADI, H. M. J.; AL-AZZAWI, I. A. A survey on automotive cybersecurity: Vulnerabilities, attacks, and countermeasures. In: IEEE. *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*. [S.l.], 2021. p. 216–221.

APÊNDICES

I. HEADER DA BIBLIOTECA `CAN_ESP_LIB`, CONTENDO AS DEFINIÇÕES DAS FUNÇÕES BÁSICAS DA API

```
1  // ============================================
2  // ARQUIVO: can_esp_lib.h
3  // Biblioteca principal CAN-ESP
4  // ============================================
5
6  #ifndef CAN_ESP_LIB_H
7  #define CAN_ESP_LIB_H
8
9  #include "freertos/FreeRTOS.h"
10 #include "freertos/queue.h"
11 #include "freertos/task.h"
12 #include "freertos/semphr.h"
13 #include "driver/twai.h"
14 #include "esp_system.h"
15 #include "esp_log.h"
16 #include "esp_ota_ops.h"
17 #include "esp_http_client.h"
18 #include "esp_mesh.h"
19 #include "mqtt_client.h"
20 #include "cJSON.h"
21
22 #ifdef __cplusplus
23 extern "C" {
24 #endif
25
26 // ============================================
27 // CONSTANTES E DEFINIÇÕES GERAIS
28 // ============================================
29
30 #define CAN_ESP_VERSION_MAJOR 1
31 #define CAN_ESP_VERSION_MINOR 0
32 #define CAN_ESP_VERSION_PATCH 0
33
34 #define CAN_ESP_LIB_TAG "CAN_ESP"
35
```

```c
// Taxas de transmissão suportadas (conforme Capítulo 2)
typedef enum {
    CAN_ESP_BITRATE_1M = 1000000,
    CAN_ESP_BITRATE_500K = 500000,
    CAN_ESP_BITRATE_250K = 250000,
    CAN_ESP_BITRATE_125K = 125000,
    CAN_ESP_BITRATE_100K = 100000,
    CAN_ESP_BITRATE_50K = 50000
} can_esp_bitrate_t;

// Formatos de frame (Seção 2.3)
typedef enum {
    CAN_ESP_FRAME_STANDARD, // 11-bit identifier
    CAN_ESP_FRAME_EXTENDED // 29-bit identifier (padrão CAN-ESP)
} can_esp_frame_format_t;

// Prioridades de mensagem (Tabela 4.2)
typedef enum {
    CAN_ESP_PRIORITY_CRITICAL = 0, // Nível 1: Freio (ID 0x001)
    CAN_ESP_PRIORITY_HIGH = 1, // Nível 2: Aceleração (ID 0x002)
    CAN_ESP_PRIORITY_MEDIUM = 2, // Nível 3: Direção (ID 0x003)
    CAN_ESP_PRIORITY_LOW = 3, // Nível 4: Telemetria
    CAN_ESP_PRIORITY_BACKGROUND = 4 // Nível 5: Diagnóstico
} can_esp_priority_t;

// Estados da rede (Figura 4.8 e Tabela 4.4)
typedef enum {
    CAN_ESP_STATE_IDLE,
    CAN_ESP_STATE_OPERATIONAL,
    CAN_ESP_STATE_OTA_DOWNLOADING,
    CAN_ESP_STATE_OTA_DISTRIBUTING,
    CAN_ESP_STATE_OTA_INSTALLING,
    CAN_ESP_STATE_OTA_VALIDATING,
    CAN_ESP_STATE_ERROR
} can_esp_state_t;

// Tipos de nó na rede mesh (Capítulo 4.5.1)
typedef enum {
    CAN_ESP_NODE_ROOT, // Nó raiz (gateway OTA)
    CAN_ESP_NODE_INTERMEDIATE, // Nó intermediário
    CAN_ESP_NODE_LEAF // Nó folha
} can_esp_node_type_t;
```

```
78
79 // ============================================
80 // ESTRUTURAS DE DADOS PRINCIPAIS
81 // ============================================
82
83 // Estrutura de mensagem CAN (Seção 2.3)
84 typedef struct {
85     uint32_t id; // Identificador (11 ou 29 bits)
86     uint8_t data[8]; // Payload (0-8 bytes)
87     uint8_t data_length_code; // DLC (0-8)
88     can_esp_frame_format_t format; // Formato do frame
89     uint64_t timestamp_tx;
90     uint64_t timestamp_rx;
91     can_esp_priority_t priority; // Prioridade da aplicação
92     uint16_t sequence_number; // Número de sequência para controle
93     bool requires_ack; // Requer confirmação explícita
94 } can_esp_message_t;
95
96 // Estrutura de configuração da rede (Capítulo 4)
97 typedef struct {
98     can_esp_bitrate_t bitrate; // Taxa de transmissão (1 Mbps padrão)
99     uint8_t tx_pin; // Pino TX (GPIO_NUM_X)
100    uint8_t rx_pin; // Pino RX (GPIO_NUM_X)
101    uint32_t node_id; // ID único do nó (0-127)
102    can_esp_node_type_t node_type; // Tipo do nó na rede
103    uint32_t queue_size_tx; // Tamanho da fila de transmissão
104    uint32_t queue_size_rx; // Tamanho da fila de recepção
105    bool enable_timestamping; // Habilita timestamp de alta precisão
106    bool enable_error_counters; // Habilita contadores de erro
107    bool enable_performance_metrics; // Habilita métricas de
       ↪ desempenho
108 } can_esp_config_t;
109
110 // Estrutura de métricas de desempenho (Capítulo 5)
111 typedef struct {
112    // Métricas de tráfego
113    uint32_t total_frames_tx;
114    uint32_t total_frames_rx;
115    uint32_t total_bytes_tx;
116    uint32_t total_bytes_rx;
117
118    // Métricas de tempo real (Seção 5.2)
```

```
119     float bus_load_percentage; // Carga do barramento (%)
120     uint32_t arbitration_losses; // Perdas de arbitragem
121     uint32_t retransmission_count; // Contador de retransmissões
122     uint32_t error_frames; // Frames de erro detectados
123 
124     // Latências (Seções 5.2.5-5.2.8)
125     float average_latency_us; // Latência média
126     float worst_case_latency_us; // WCRT observado
127     float average_jitter_us; // Jitter médio
128     float max_jitter_us; // Jitter máximo
129 
130     // Contadores de erro (Seção 5.2.3)
131     uint32_t transmit_error_counter;
132     uint32_t receive_error_counter;
133     uint8_t error_state; // Error Active/Passive/Bus-Off
134 
135     // Timestamps
136     uint64_t measurement_start;
137     uint64_t measurement_end;
138 } can_esp_performance_metrics_t;
139 
140 // Estrutura para configuração OTA (Capítulo 4.5)
141 typedef struct {
142     char mqtt_broker_url[128]; // URL do broker MQTT
143     char mqtt_topic_command[64]; // Tópico para comandos
144     char mqtt_topic_status[64]; // Tópico para status
145     char https_firmware_url[256]; // URL para download do firmware
146     char mesh_ssid[32]; // SSID da rede mesh
147     char mesh_password[64]; // Senha da rede mesh
148     uint8_t mesh_channel; // Canal Wi-Fi
149     uint16_t mesh_port; // Porta para comunicação mesh
150     uint32_t chunk_size; // Tamanho do chunk (1024 bytes padrão)
151     uint8_t max_retries; // Máximo de tentativas
152     uint32_t timeout_ms; // Timeout em milissegundos
153 } can_esp_ota_config_t;
154 
155 // Estrutura do estado OTA (Tabela 4.4)
156 typedef struct {
157     can_esp_state_t state;
158     uint32_t target_node_id;
159     uint32_t firmware_size;
160     uint32_t bytes_transferred;
```

```
161     uint32_t chunks_transferred;
162     float progress_percentage;
163     uint8_t retry_count;
164     esp_err_t last_error;
165     char firmware_version[32];
166     uint8_t firmware_hash[32]; // SHA-256 do firmware
167 } can_esp_ota_status_t;
168 
169 // ============================================
170 // PROTÓTIPOS DE FUNÇÕES PRINCIPAIS
171 // ============================================
172 
173 // ---------------------------------------------------------------
174 // SEÇÃO 1: INICIALIZAÇÃO E CONTROLE DA REDE
175 // ---------------------------------------------------------------
176 
177 /**
178  * @brief Inicializa a biblioteca CAN-ESP
179  * @param config Ponteiro para estrutura de configuração
180  * @return ESP_OK em caso de sucesso, código de erro em caso de
       ↪ falha
181  *
182  * Descrição: Implementa a função CAN_ESP_InitWithConfig()
       ↪ mencionada
183  * na Seção 4.4.1, encapsulando toda a complexidade de inicialização
       ↪ .
184  */
185 esp_err_t can_esp_init(const can_esp_config_t *config);
186 
187 /**
188  * @brief Inicia a operação da rede CAN
189  * @return ESP_OK em caso de sucesso
190  */
191 esp_err_t can_esp_start(void);
192 
193 /**
194  * @brief Para a operação da rede CAN
195  * @return ESP_OK em caso de sucesso
196  */
197 esp_err_t can_esp_stop(void);
198 
199 /**
```

```
200  * @brief Desinicializa a biblioteca CAN-ESP
201  * @return ESP_OK em caso de sucesso
202  */
203 esp_err_t can_esp_deinit(void);
204
205 // ---------------------------------------------------------------------
206 // SEÇÃO 2: TRANSMISSÃO E RECEPÇÃO DE MENSAGENS
207 // ---------------------------------------------------------------------
208
209 /**
210  * @brief Enfileira uma mensagem para transmissão
211  * @param message Ponteiro para a mensagem a ser transmitida
212  * @param timeout_ticks Timeout em ticks do FreeRTOS
213  * @return ESP_OK em caso de sucesso
214  *
215  * Descrição: Implementa a função CAN_ESP_EnqueueMessage()
       ↪ mencionada
216  * na Seção 4.4.1, usando padrão produtor-consumidor.
217  */
218 esp_err_t can_esp_enqueue_message(const can_esp_message_t *message,
219                                   TickType_t timeout_ticks);
220
221 /**
222  * @brief Recebe uma mensagem da fila de recepção
223  * @param message Ponteiro para armazenar a mensagem recebida
224  * @param timeout_ticks Timeout em ticks do FreeRTOS
225  * @return ESP_OK em caso de sucesso, ESP_ERR_TIMEOUT se timeout
226  */
227 esp_err_t can_esp_receive_message(can_esp_message_t *message,
228                                   TickType_t timeout_ticks);
229
230 /**
231  * @brief Envia uma mensagem de forma síncrona
232  * @param message Ponteiro para a mensagem a ser enviada
233  * @return ESP_OK em caso de sucesso
234  */
235 esp_err_t can_esp_send_message(const can_esp_message_t *message);
236
237 /**
238  * @brief Configura filtros de recepção
239  * @param filter_mask Máscara para filtragem
240  * @param filter_code Código para filtragem
```

```
241  * @return ESP_OK em caso de sucesso
242  *
243  * Descrição: Implementa filtragem conforme Figura 2.6
244  */
245 esp_err_t can_esp_set_receive_filter(uint32_t filter_mask,
246                                      uint32_t filter_code);
247
248 // ---------------------------------------------------------------
249 // SEÇÃO 3: GERENCIAMENTO DE ESTADO E DIAGNÓSTICO
250 // ---------------------------------------------------------------
251
252 /**
253  * @brief Obtém o estado atual da rede
254  * @param state Ponteiro para armazenar o estado
255  * @return ESP_OK em caso de sucesso
256  */
257 esp_err_t can_esp_get_state(can_esp_state_t *state);
258
259 /**
260  * @brief Obtém métricas de desempenho atuais
261  * @param metrics Ponteiro para armazenar as métricas
262  * @return ESP_OK em caso de sucesso
263  */
264 esp_err_t can_esp_get_performance_metrics(
      ↪ can_esp_performance_metrics_t *metrics);
265
266 /**
267  * @brief Reseta as métricas de desempenho
268  * @return ESP_OK em caso de sucesso
269  */
270 esp_err_t can_esp_reset_metrics(void);
271
272 /**
273  * @brief Obtém os contadores de erro do controlador TWAi
274  * @param tec Ponteiro para contador de erros de transmissão
275  * @param rec Ponteiro para contador de erros de recepção
276  * @return ESP_OK em caso de sucesso
277  *
278  * Descrição: Implementa monitoramento conforme Seção 5.2.3
279  */
280 esp_err_t can_esp_get_error_counters(uint32_t *tec, uint32_t *rec);
281
```

```
282 /**
283  * @brief Verifica integridade da comunicação
284  * @return true se comunicação íntegra, false caso contrário
285  */
286 bool can_esp_check_communication_integrity(void);
287 
288 // ---------------------------------------------------------------
289 // SEÇÃO 4: SISTEMA DE ATUALIZAÇÃO OTA
290 // ---------------------------------------------------------------
291 
292 /**
293  * @brief Inicializa o subsistema OTA
294  * @param config Ponteiro para configuração OTA
295  * @return ESP_OK em caso de sucesso
296  */
297 esp_err_t can_esp_ota_init(const can_esp_ota_config_t *config);
298 
299 /**
300  * @brief Inicia processo de atualização OTA para um nó específico
301  * @param target_node_id ID do nó alvo
302  * @param firmware_url URL do firmware (opcional, usa config se NULL
      ↪ )
303  * @return ESP_OK em caso de sucesso
304  */
305 esp_err_t can_esp_ota_start_update(uint32_t target_node_id,
306                                    const char *firmware_url);
307 
308 /**
309  * @brief Obtém status do processo OTA em andamento
310  * @param status Ponteiro para armazenar status
311  * @return ESP_OK em caso de sucesso
312  */
313 esp_err_t can_esp_ota_get_status(can_esp_ota_status_t *status);
314 
315 /**
316  * @brief Cancela processo de atualização OTA em andamento
317  * @return ESP_OK em caso de sucesso
318  */
319 esp_err_t can_esp_ota_cancel_update(void);
320 
321 /**
322  * @brief Verifica se o sistema está em estado seguro para OTA
```

```
323  * @return true se seguro para atualização, false caso contrário
324  *
325  * Descrição: Implementa Safe State Check conforme Seção 4.5.3
326  */
327 bool can_esp_ota_is_safe_state(void);
328 
329 /**
330  * @brief Registra callback para notificações OTA
331  * @param callback Função de callback
332  * @param arg Argumento para callback
333  * @return ESP_OK em caso de sucesso
334  */
335 esp_err_t can_esp_ota_register_callback(void (*callback)(
      ↪ can_esp_ota_status_t *, void *),
336                                        void *arg);
337 
338 // ---------------------------------------------------------------
339 // SEÇÃO 5: REDE MESH E COMUNICAÇÃO DISTRIBUÍDA
340 // ---------------------------------------------------------------
341 
342 /**
343  * @brief Inicializa a rede mesh Wi-Fi
344  * @param ssid SSID da rede mesh
345  * @param password Senha da rede mesh
346  * @param channel Canal Wi-Fi
347  * @return ESP_OK em caso de sucesso
348  */
349 esp_err_t can_esp_mesh_init(const char *ssid, const char *password,
      ↪ uint8_t channel);
350 
351 /**
352  * @brief Envia dados através da rede mesh
353  * @param dest_node_id ID do nó destino
354  * @param data Dados a serem enviados
355  * @param size Tamanho dos dados
356  * @return ESP_OK em caso de sucesso
357  */
358 esp_err_t can_esp_mesh_send(uint32_t dest_node_id, const uint8_t *
      ↪ data, size_t size);
359 
360 /**
361  * @brief Recebe dados da rede mesh
```

```
362  * @param data Buffer para armazenar dados
363  * @param size Tamanho do buffer
364  * @param source_node_id Ponteiro para armazenar ID de origem
365  * @param timeout_ticks Timeout em ticks
366  * @return Número de bytes recebidos
367  */
368 size_t can_esp_mesh_receive(uint8_t *data, size_t size,
369                             uint32_t *source_node_id,
370                             TickType_t timeout_ticks);
371
372 /**
373  * @brief Obtém topologia atual da rede mesh
374  * @param topology Buffer para armazenar topologia (JSON)
375  * @param buffer_size Tamanho do buffer
376  * @return ESP_OK em caso de sucesso
377  */
378 esp_err_t can_esp_mesh_get_topology(char *topology, size_t
      ↪ buffer_size);
379
380 // ---------------------------------------------------------------
381 // SEÇÃO 6: UTILITÁRIOS E FUNÇÕES AUXILIARES
382 // ---------------------------------------------------------------
383
384 /**
385  * @brief Converte ID CAN para string hexadecimal
386  * @param id ID CAN
387  * @param format Formato do frame
388  * @param buffer Buffer para string
389  * @param buffer_size Tamanho do buffer
390  */
391 void can_esp_id_to_string(uint32_t id, can_esp_frame_format_t format
      ↪ ,
392                           char *buffer, size_t buffer_size);
393
394 /**
395  * @brief Converte dados CAN para string hexadecimal
396  * @param data Dados CAN
397  * @param length Comprimento dos dados
398  * @param buffer Buffer para string
399  * @param buffer_size Tamanho do buffer
400  */
401 void can_esp_data_to_string(const uint8_t *data, uint8_t length,
```

```
402                     char *buffer, size_t buffer_size);
403
404 /**
405  * @brief Calcula CRC32 para verificação de integridade
406  * @param data Dados para cálculo
407  * @param length Comprimento dos dados
408  * @return Valor CRC32
409  */
410 uint32_t can_esp_calculate_crc32(const uint8_t *data, size_t length)
      ↪ ;
411
412 /**
413  * @brief Obtém timestamp de microssegundos de alta precisão
414  * @return Timestamp em microssegundos
415  */
416 uint64_t can_esp_get_timestamp_us(void);
417
418 /**
419  * @brief Gera relatório de diagnóstico em formato JSON
420  * @param json_buffer Buffer para JSON
421  * @param buffer_size Tamanho do buffer
422  * @return ESP_OK em caso de sucesso
423  */
424 esp_err_t can_esp_generate_diagnostic_report(char *json_buffer,
      ↪ size_t buffer_size);
425
426 // ---------------------------------------------------------------
427 // SEÇÃO 7: CALLBACKS E HANDLERS
428 // ---------------------------------------------------------------
429
430 // Tipo de callback para mensagens recebidas
431 typedef void (*can_esp_message_callback_t)(const can_esp_message_t *
      ↪ message, void *arg);
432
433 /**
434  * @brief Registra callback para mensagens recebidas
435  * @param callback Função de callback
436  * @param arg Argumento para callback
437  * @return ESP_OK em caso de sucesso
438  */
439 esp_err_t can_esp_register_message_callback(
      ↪ can_esp_message_callback_t callback, void *arg);
```

```
440
441 /**
442  * @brief Registra callback para eventos de erro
443  * @param callback Função de callback
444  * @param arg Argumento para callback
445  * @return ESP_OK em caso de sucesso
446  */
447 esp_err_t can_esp_register_error_callback(void (*callback)(esp_err_t
      ↪  error, void *arg), void *arg);
448
449 // ---------------------------------------------------------------
450 // SEÇÃO 8: FUNÇÕES ESPECÍFICAS DO CAN-ESP (Protocolo da Aplicação)
451 // ---------------------------------------------------------------
452
453 /**
454  * @brief Envia comando de frenagem de emergência
455  * @param pressure Pressão de frenagem (0-100%)
456  * @param emergency true para frenagem de emergência
457  * @return ESP_OK em caso de sucesso
458  *
459  * Descrição: Implementa mensagem ID 0x001 conforme Tabela 4.2
460  */
461 esp_err_t can_esp_send_brake_command(float pressure, bool emergency)
      ↪ ;
462
463 /**
464  * @brief Envia comando de aceleração
465  * @param throttle_position Posição do acelerador (0-100%)
466  * @return ESP_OK em caso de sucesso
467  *
468  * Descrição: Implementa mensagem ID 0x002 conforme Tabela 4.2
469  */
470 esp_err_t can_esp_send_throttle_command(float throttle_position);
471
472 /**
473  * @brief Envia comando de direção
474  * @param steering_angle Ângulo de direção (-100 a +100%)
475  * @return ESP_OK em caso de sucesso
476  *
477  * Descrição: Implementa mensagem ID 0x003 conforme Tabela 4.2
478  */
479 esp_err_t can_esp_send_steering_command(float steering_angle);
```

```
480
481 /**
482  * @brief Envia telemetria do motor
483  * @param rpm Rotação por minuto
484  * @param temperature Temperatura do motor (Celcius)
485  * @param current Corrente do motor (A)
486  * @return ESP_OK em caso de sucesso
487  */
488 esp_err_t can_esp_send_motor_telemetry(float rpm, float temperature,
      ↪  float current);
489
490 /**
491  * @brief Envia status da bateria
492  * @param voltage Tensão da bateria (V)
493  * @param current_current Corrente da bateria (A)
494  * @param state_of_charge Estado de carga (%)
495  * @param temperature Temperatura da bateria (Celcius)
496  * @return ESP_OK em caso de sucesso
497  */
498 esp_err_t can_esp_send_battery_status(float voltage, float
      ↪ current_current,
499                                       float state_of_charge, float
                                             ↪ temperature);
500
501 // ---------------------------------------------------------------------
502 // SEÇÃO 9: ANÁLISE DE TEMPO REAL (WCRT E JITTER)
503 // ---------------------------------------------------------------------
504
505 /**
506  * @brief Calcula WCRT teórico para uma mensagem específica
507  * @param message_id ID da mensagem
508  * @param priority Prioridade da mensagem
509  * @param bus_load Carga do barramento (%)
510  * @return WCRT teórico em microssegundos
511  *
512  * Descrição: Implementa cálculo baseado em Tindell et al. (1995)
513  * conforme Equações 2.1 e 2.2 na Seção 2.1.3
514  */
515 float can_esp_calculate_theoretical_wcrt(uint32_t message_id,
516                                          can_esp_priority_t priority,
517                                          float bus_load);
518
```

```
519 /**
520  * @brief Mede jitter real da comunicação
521  * @param message_id ID da mensagem para análise
522  * @param sample_count Número de amostras
523  * @param avg_jitter Ponteiro para jitter médio
524  * @param max_jitter Ponteiro para jitter máximo
525  * @return ESP_OK em caso de sucesso
526  */
527 esp_err_t can_esp_measure_jitter(uint32_t message_id, uint32_t
      ↪ sample_count,
528                                  float *avg_jitter, float *max_jitter);
529
530 /**
531  * @brief Analisa escalabilidade do sistema
532  * @param node_count Número de nós hipotético
533  * @param bus_load Carga do barramento alvo (%)
534  * @param results Buffer para resultados (JSON)
535  * @param buffer_size Tamanho do buffer
536  * @return ESP_OK em caso de sucesso
537  *
538  * Descrição: Implementa modelagem teórica conforme Seção 4.5.6
539  */
540 esp_err_t can_esp_analyze_scalability(uint32_t node_count, float
      ↪ bus_load,
541                                       char *results, size_t buffer_size);
542
543 // ---------------------------------------------------------------------
544 // SEÇÃO 10: CONFIGURAÇÃO E PARÂMETROS DO SISTEMA
545 // ---------------------------------------------------------------------
546
547 /**
548  * @brief Obtém versão da biblioteca
549  * @param major Ponteiro para versão major
550  * @param minor Ponteiro para versão minor
551  * @param patch Ponteiro para versão patch
552  */
553 void can_esp_get_version(uint8_t *major, uint8_t *minor, uint8_t *
      ↪ patch);
554
555 /**
556  * @brief Configura timeout de transmissão
557  * @param timeout_ms Timeout em milissegundos
```

```
558  * @return ESP_OK em caso de sucesso
559  */
560 esp_err_t can_esp_set_transmit_timeout(uint32_t timeout_ms);
561
562 /**
563  * @brief Configura timeout de recepção
564  * @param timeout_ms Timeout em milissegundos
565  * @return ESP_OK em caso de sucesso
566  */
567 esp_err_t can_esp_set_receive_timeout(uint32_t timeout_ms);
568
569 /**
570  * @brief Habilita/desabilita modo de baixa potência
571  * @param enable true para habilitar, false para desabilitar
572  * @return ESP_OK em caso de sucesso
573  */
574 esp_err_t can_esp_set_low_power_mode(bool enable);
575
576 #ifdef __cplusplus
577 }
578 #endif
579
580 #endif // CAN_ESP_LIB_H
```