MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

MULTIPLEXAÇÃO DE PACOTES CAN EM QUADROS ETHERNET

Renan Airton Batista Ribeiro

Orientador: Prof. Me. Everthon Valadão

FORMIGA- MG
2017

RENAN AIRTON BATISTA RIBEIRO

MULTIPLEXAÇÃO DE PACOTES CAN EM QUADROS ETHERNET

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Everthon Valadão.

FORMIGA- MG
2017

Ribeiro, Renan Airton Batista.
004 Multiplexação de pacotes can em quadros ethernet / Renan Airton
R484m Batista Ribeiro. – Formiga : IFMG, 2017.
89p. : il.

Orientador: Prof. Me. Everthon Valadão dos Santos.
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. TSN. 2. Enfileiramento. 3. CANbus. 4. Ethernet. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Me.. Naliana Dias Leandro CRB6-1347

Agradecimentos

A Deus por ter me dado força nos momentos mais difíceis. Ao meu pai e minha tia Maria Geralda que sempre me apoiaram e tornaram este trabalho possível. A minha mãe que sempre me mostrou o valor da educação e me impulsionou aos estudos. Ao meu orientador Everthon Valadão que despertou em mim o interesse por este trabalho, pelo incentivo e ajuda durante todo o desenvolvimento do mesmo. A todos professores do IFMG que contribuíram não só para o meu conhecimento acadêmico mas também para meu caráter e formação profissional. A todas as pessoas que possam estar lendo estas palavras e sabem que foram fundamentais na minha trajetória até aqui. E a todos que direta ou indiretamente contribuíram para minha formação, o meu muito obrigado.

Resumo

À medida que a indústria automotiva agrega novos sensores e atuadores aos veículos, cresce a necessidade de aumento da vazão de dados na rede intraveicular. Para sanar essa limitação, a indústria automotiva tem investido esforços consideráveis na adaptação da tecnologia Ethernet para aplicações em redes veiculares (KERN, 2012). Há um considerável esforço em transformar a tecnologia Ethernet na rede *backbone* intraveicular, aposentando gradualmente a rede CAN e complementares (tecnologia presente nos veículos atuais). A proposta deste trabalho consiste no desenvolvimento e implementação de um algoritmo com estratégia(s) de multiplexação de quadros CAN em pacotes Ethernet, com o objetivo de criar uma ponte entre essas duas tecnologias, auxiliando na transição do legado das redes veiculares. Este tema possui grande relevância no cenário automobilístico atual, onde a demanda por maior vazão de dados em redes veiculares é crescente. As principais contribuições deste trabalho são: dentre as várias estratégias implementadas, a que obteve o segundo melhor uso do canal Ethernet é uma das propostas deste trabalho, Gatilho Estocástico por Prioridade, alcançando um uso efetivo de 97% do canal Ethernet, com um atraso 1,4 vezes menor que o melhor caso (100% de uso efetivo). Já a estratégia Gatilho Estocástico por Ciclo, também proposta neste trabalho, apresentou o menor atraso médio por CAN ID bem como garantiu o menor atraso máximo absoluto dentre todas as demais estratégias de enfileiramento simuladas. Quanto à aplicação dos resultados deste trabalho, o algoritmo Gatilho Estocástico por Ciclo poderia ser utilizado por aplicações veiculares que tolerem um atraso médio da ordem de 300 μs, ao passo que garantiria (no cenário com 100% de carga) o uso efetivo de 76% da capacitação do canal Ethernet. Caso a aplicação veicular seja tolerante a atrasos da ordem de milissegundos, ou necessite de maior taxa efetiva de dados, o algoritmo Gatilho Estocástico por Prioridade garantiria (no mesmo cenário) uma taxa efetiva de 97 Mbps numa rede Ethernet de 100 Mbps, só não sendo mais eficiente do que o melhor caso (fila sempre cheia), ao custo de um atraso médio por CAN ID de 3,4 ms. Todos os *scripts* e códigos-fontes implementados neste trabalho foram disponibilizadas via GitHub[1].

Palavras-chave: TSN. Enfileiramento. CANbus. Multiplexação. Ethernet.

[1] https://github.com/ribeirorenan/tcc-multiplexacao-can-ethernet

Abstract

Nowadays, the automotive industry aggregates new sensors and actuators to vehicles. It creates the need for a higher data throughput in the automotive network. To get over this problem, the automotive industry has invested heavily in the adaptation of Ethernet technology for use in vehicular networks (KERN, 2012). There is considerable effort to transform Ethernet technology into the vehicular backbone network, and gradually retires the CAN network (omnipresent technology in today's vehicles). The purpose of this work is to develop and implement an algorithm that allow multiplexing CAN frame in Ethernet packets, with the purpose of creating a bridge between these two technologies. This topic has great relevance in the current automotive scenario, where the need for higher throughput in vehicular networks is increasing. The main contributions of this work are: among the implemented strategies, the one that obtained the second best use of the Ethernet throughput is one of the proposals of this work, Stochastic Trigger by Priority, achieving an effective use of 97% of the Ethernet throughput, but with a delay 1.4 times lower than the best case, which would be the maximum use of the payload Ethernet. The strategy Stochastic Trigger by Cycle (also proposed in this work) presented the lowest average delay by CAN ID as well as guaranteed the lowest absolute maximum delay among all other simulated queuing strategies. As for the application of the results of this work, the algorithm Stochastic Trigger by Cycle could be used by vehicular applications that tolerate an average delay of the order of 300 μs, while guaranteeing (in the scenario with 100% busload) a 76% of effective utilization of the Ethernet throughput, that is, 76 Mbps effective on a nominal 100 Mbps Ethernet network. If the vehicular application is tolerant to delays of the order of milliseconds, or requires a higher effective data rate, the algorithm Stochastic Trigger by Priority guaranteed (in the same scenario) an effective rate of 97 Mbps in a 100 Mbps Ethernet network, only being less efficient than the best case, that dispatch the queue always full, at the cost of an average delay for CAN ID of 3.4 ms and an absolute delay of at most 10.4 ms for 99% of the simulated CAN messages. All scripts and source-codes implemented in this work are available via GitHub [2]

Keywords: TSN. Queueing. CANbus. Multiplexing. Ethernet.

[2] https://github.com/ribeirorenan/tcc-multiplexacao-can-ethernet

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

ADAS	Advanced Driver-Assistance Systems
AFDX	*Avionics Full-Duplex Switched Ethernet*
AVB	*Audio Video Bridging*
CAN	*Controller Area Network*
CAN-FD	*Controller Area Network with Flexible Data-Rate*
CSMA/CD	*Carrier-Sense Multiple Access with Collision Detection*
DSP	*Digital Signal Processing*
EOF	*End Of Frame*
IHM	Interface Homem Máquina
IEC	*International Electrotechnical Commission*
ISA	*Industry Standard Architecture*
MAC	*Media Access Control*
MOST	*Media Oriented Systems Transport*
PAM-3	*Pulse Amplitude Modulation 3*
PLC	*Programmable Logic Controller*
SAE	*Society of Automotive Engineers*
SBC	Sociedade Brasileira de Computação
TCC	Trabalho de Conclusão de Curso
TDMA	*Time-Division Multiple Access*
TSN	*Time-Sensitive Networking*
OSI	*Open Systems Interconnection*
WCRT	*Worst Case Response Time*

Sumário

1 INTRODUÇÃO

Na arquitetura de rede veicular, existem basicamente quatro grandes áreas de aplicação: **propulsão** (motor e câmbio); **dinâmica** (freio e direção automáticos, modo de cruzeiro); **carroceria** (iluminação, climatização, travas, vidros e espelhos); **multimídia** (navegação e conectividade). Cada uma dessas funções apresenta necessidades particulares, que envolvem a largura de banda suportada pela rede, tempo de resposta e susceptibilidade a erros (SOUZA; NERY; CAMPOS, 2016). A tradicional rede CAN tem um desempenho satisfatório para a maioria dessas áreas de aplicação, como processamento rápido para funções de propulsão e dinâmica. Funções de carroceria necessitam de uma interface direta com o motorista que muitas vezes não necessariamente devem ser processados em altas velocidades, respeitando apenas o tempo de resposta humana (SOUZA; NERY; CAMPOS, 2016). O barramento serial Controller Area Network (CAN) foi desenvolvido pela Bosch em meados da década de 1980 (FREDRIKSSON, 1996), para prover um sistema de comunicação automotiva com excelente custo benefício. Atualmente, grande parte dos automóveis ainda tem seu sistema de comunicação intraveicular baseado no barramento CAN.

Nas últimas décadas é notável a tendência onde fabricantes de veículos automotores tem buscado proporcionar mais conforto e tecnologia para os motoristas (LOUREIRO, 2013). Tradicionalmente, os veículos utilizavam poucas dezenas de centrais eletrônicas – ECUs, trocando poucos milhares de mensagens sobre seus sensores e atuadores através da rede CAN. Entretanto, na atualidade a competição do mercado automobilístico, em relação aos veículos populares, já não se diferencia apenas pelo *design* e pela potência que o motor proporciona, mas também pelos "atrativos" que já se tornam praticamente obrigatórios em modelos novos (SEDGWICK, 2014). Afinal, carros estão se tornando "computadores sobre rodas" e isso torna evidente a importância de redes veiculares com melhor desempenho para suportar todos estes recém adotados "atrativos" tecnológicos. Na maioria dos modelos de luxo e em alguns intermediários recentemente lançados, os veículos já oferecem sistemas de auxílio ao motorista (ADAS), recursos avançados de multimídia, telemetria e IHM (interface homem-máquina), para os quais a rede CAN sozinha não consegue atender aos requisitos de taxa de dados. A taxa de transmissão limitada oferecida pela CAN não é suficiente para atender à crescente demanda na troca de dados entre as atuais centenas de ECUs, ensejando enviar milhões de mensagens sobre seus sensores e atuadores.

Conforme observado, à medida que a indústria automotiva agrega novos sensores e atuadores, cresce a necessidade de aumento da vazão de dados na rede intraveicular. Para

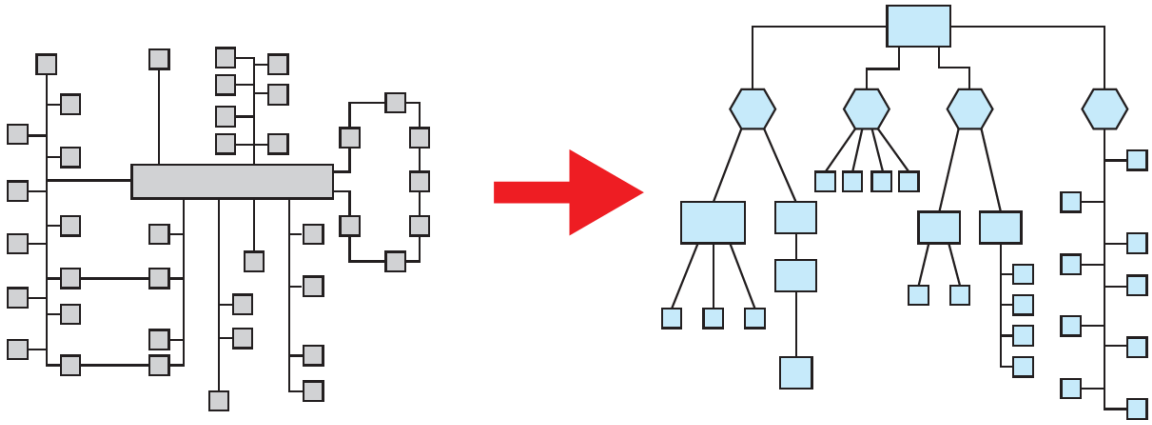
sanar essa limitação, a indústria automotiva tem investido esforços consideráveis na adaptação da tecnologia Ethernet para aplicações em redes veiculares (KERN, 2012). Há um considerável esforço da indústria em transformar a tecnologia Ethernet na rede *backbone* intraveicular, aposentado gradualmente a rede CAN e complementares. Graças às altas vazões proporcionadas pela Ethernet, funções avançadas de assistência ao motorista (ADAS) e de *infotainment* seriam de adoção mais viável, mesmo em carros populares (SOUZA; NERY; CAMPOS, 2016). A iniciativa da indústria pelo desenvolvimento e adaptação da rede Ethernet em aplicações veiculares resultou na criação dos consórcios *OPEN Alliance*[1] e *AVnu Alliance*[2], que vêm trabalhando em cooperação com a IEEE com o objetivo de resolverem os principais desafios hoje encontrados na adoção da rede Ethernet em aplicações automotivas. Desta cooperação, surgiram padronizações para enlaces Ethernet de 100 Mbit/s (IEEE 802.3bw, 2015) e 1000 Mbit/s (IEEE 802.3bp, 2016) utilizando apenas um par de fios trançados, bem como padrões para aplicações com sensibilidade temporal (latências ultra baixas e determinísticas) como os protocolos AVB ((IEEE 802.1BA, 2014) (IEEE 802.1AS, 2013) (IEEE 802.1Qat, 2010) (IEEE 802.Qav, 2009)).

Entretanto, apesar de desejada, uma mudança brusca na infraestrutura das redes intraveiculares não é viável economicamente, sendo necessário um processo de transição gradual entre tecnologias. Deve-se ser capaz de interconectar a tradicional rede intraveicular CAN (utilizada desde a década de 70), que transporta informações urgentes e essenciais (e.g. velocidade, rotação do motor, consumo, avisos de falha), com as novas redes veiculares que dão suporte aos recém integrados sistemas de conforto (e.g. centrais multimídias, *infotainment*) e ADAS — assistência avançada ao motorista (JUNIOR, 2012). Até que haja uma adoção universal e homogênea da Ethernet veicular, a solução é o uso de sistemas híbridos, com duas ou mais redes empregadas em funções específicas (e.g. missão crítica e sistemas de conforto) e o uso de *gateways* para a transmissão de informações pertinentes entre elas (*bridging*), tendo a Ethernet veicular ("tronco" na topologia em árvore) o papel de *backbone* entre estas redes ("ramificações"), como ilustra a Figura 1.

Pelas altas taxas de dados oferecidas, a Ethernet Veicular inicialmente seria utilizada como *backbone* entre as redes intra veiculares, tendo o concentrador Ethernet o papel de interconexão entre estes enlaces. Considerando que a rede CAN é onipresente nos veículos atuais, para transportar suas mensagens entre barramentos CAN interligados pelo concentrador Ethernet faz-se necessário a incorporação[3] dos quadros CAN em quadro Ethernet. Porém, se comparado ao CAN, o quadro Ethernet oferece uma carga de dados útil mais de 100 vezes maior que um quadro CAN, com cerca de 1500 bytes de carga útil transmitidos a taxas de 100 Mbit/s a nível de enlace. A rede Ethernet apresenta a possibilidade de se transmitir mensagens com diferentes tamanhos de quadros de dados

[1] http://www.opensig.org/members/
[2] http://avnu.org/our-members/
[3] incorporar, no sentido de: encapsulamento, tunelamento, multiplexação.

Figura 1 – Mudança de arquitetura veicular: de barramentos para árvore ramificada.



Fonte: (IXIA, 2014)

de carga útil (de 46 a 1500 bytes), o que garante uma comunicação mais eficiente. Já um quadro CAN 2.0A pode ter de 5,5 a 13,5 bytes e CAN 2.0B de 8 a 16 *bytes*, ou seja, no *payload* de um quadro Ethernet cabem no mínimo 2 e no máximo 272 quadros CAN. Uma estratégia de multiplexação 1-1 desperdiçaria a carga útil Ethernet, enquanto que uma estratégia N-1 poderia fazer com que mensagens CAN prioritárias não cumprissem seu tempo de resposta esperado (HERBER et al., 2015). Esta proposta de TCC visa a atacar este problema, onde se faz necessário um compromisso entre maximizar a taxa de utilização (*payload* ÷ *overhead*) do quadro Ethernet e a manutenção das garantias de temporização e prioridade das mensagens CAN.

1.1 Justificativa

O projeto apresentado está dentro do escopo do curso de ciência da computação, pois abrange diversas áreas, tais como: Redes de Computadores, para comunicação intraveicular, funcionamento do protocolo CAN e interconexão com Ethernet; Eletrônica Digital e Sistemas Embarcados, para manipular o *hardware* que fará o processamento dos pacotes; Sistemas Operacionais, para configuração de serviços, escalonamento de filas e alteração do comportamento de controladores de interfaces de rede. Por último, Programação, para implementação do algoritmo responsável pela multiplexação de pacotes e Estatistica para análise dos resultados da simulação. A viabilidade deste trabalho de conclusão de curso se deu pelos seguintes motivos: :

- Pertinência deste TCC à área de atuação do Polo de Inovação do IFMG (EMBRAPII) sediado no Campus Formiga, sendo ela Sistemas Automotivos Inteligentes – Aplicações Embarcadas & Protocolos de comunicação inter e extra veicular;

- Necessidade iminente da indústria automobilística (conforme esforços de consórcios como *OPEN Alliance* e *AVnu Alliance*) por melhor desempenho nas redes intraveiculares, através do aperfeiçoamento e viabilização da Ethernet veicular;[4]
- Grande área de atuação dentro do escopo do curso de Ciência da Computação (segundo a SBC), sendo observado crescente interesse acadêmico[5], atualidade e notável aumento[6] de publicações científicas sobre o tema a partir dos últimos 7 anos;
- Interesse e apreciação pessoal em relação às disciplinas envolvidas, visando aplicar os conhecimentos obtidos no curso no desenvolvimento deste TCC, bem como abertura de uma porta de entrada para o mercado de trabalho;
- Pronta disponibilidade de *softwares* e *hardwares* essenciais ao desenvolvimento do projeto.

1.2 OBJETIVOS

Desenvolver e implementar um algoritmo com estratégia(s) de multiplexação ("incorporação") de pacotes CAN dentro de quadro(s) Ethernet (encapsulamento), visando garantir baixa latência na expedição dos pacotes, de maneira a respeitar as prioridades definidas pelo identificador de mensagem CAN e os tempos de resposta esperados numa rede intraveicular.

1.2.1 Objetivos Específicos

- Aprender a transmitir dados numa rede CAN veicular, assim como conhecer o funcionamento de rede automotiva *fieldbus* e suas diferenças com Ethernet;
- Estudar abordagens para multiplexação de quadros CAN em Ethernet como, por exemplo, enfileiramento com gatilhos de tempo, escalonamento de filas, encapsulamento 1-1, N-1 e K-1;
- Implementar estratégia de multiplexação CAN-Ethernet visando utilização em pontes que interconectam estas duas tecnologias, tendo Ethernet como rede *backbone* (topologia em árvore) e CAN *bus* como redes folha (ramificações);
- Projetar simulação ou protótipo com interconexão entre redes CAN e Ethernet, devidamente configurado para utilizar a estratégia de multiplexação desenvolvida;
- Validar o mecanismo de multiplexação CAN-Ethernet em um dos seguintes cenários: simulação de tráfego entre redes CAN e Ethernet e/ou implementação e testes de protótipo de *gateway* CAN-Ethernet que interconecte dois ou mais enlaces CAN bus através de enlace Ethernet;

[4] http://embrapii.org.br/if-formiga/
[5] https://goo.gl/E9qa3j [google acadêmico]
[6] https://goo.gl/nTX8rL [wolframalpha.com]

- Conduzir experimentos e análises dos resultados para validar a temporização e priorização esperada pelas mensagens CAN, utilizando a estratégia de multiplexação desenvolvida.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Fieldbus

Durante muitos anos, a palavra "*fieldbus*" tem sido utilizada superficialmente para representar uma rede para conexão de dispositivos de campo (sensores, PLCs etc). Porém, a tecnologia *fieldbus* representa um conjunto de soluções e técnicas utilizados para problemas que são semelhantes entre si. Os requisitos do *fieldbus* começaram a ser padronizados pela IEC (*International Electrotechnical Comission*) e ISA (*International Society of Automation*), definindo o *fieldbus* da seguinte forma: um padrão de **comunicação serial e digital**, através do qual informação possa ser enviada em ambas as direções entre dispositivos de campo e os sistemas de controle, através de um **meio de comunicação compartilhado** (THOMESSE, 2005).

O *fieldbus* pode ser aplicado para diversos fins, dentre os quais pode-se citar: sistemas de produção discreta, controle de processos industriais, automação residencial, sistemas de transportes e sistemas embarcados. Alguns benefícios do *fieldbus* são:

- Diminuição dos custos de instalação;
- Facilidade de adição de componentes de campo;
- A disponibilidade de comunicação *full-duplex* entre dispositivos de campo;
- Possibilidade de implementação de estratégias de controles mais avançadas.

2.1.1 Arquitetura Fieldbus

De maneira geral, o *fieldbus* possui três camadas:

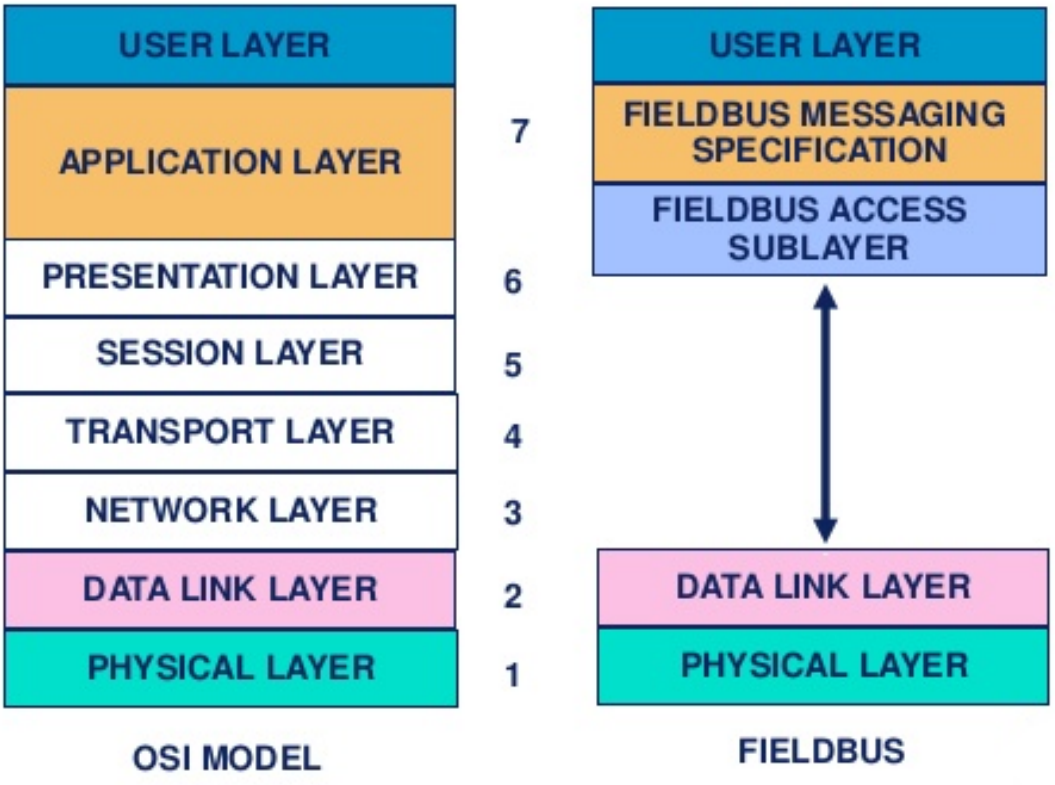
1) Camada Física;
2) Camada de Enlace;
3) Camada de Aplicação.

Se comparado ao modelo OSI, o *fieldbus* não mostra na sua divisão as camadas de rede, de transporte, de sessão e de apresentação como vemos na Figura 2 .

Entretanto, isso não quer dizer que o *fieldbus* não implemente as outras funções sugeridas pelo modelo OSI e sim que sua implementação difere do sugerido pelo modelo OSI (tendo em vista o modelo OSI é apenas conceitual). A camada de rede não é necessária no *fieldbus*, pois, geralmente apenas um único caminho é o suficiente para a rede e em sistemas mais complexos podem ser utilizadas pontes para a interconexão dos barramentos

fieldbus. Assim como a camada de rede, a camada de sessão tipicamente também não possui papel no *fieldbus* devido à natureza do mesmo. Já a função da camada de transporte, é realizada pela camada de enlace e a camada de aplicação normalmente realiza a função da camada de apresentação (THOMESSE, 2005).

Figura 2 – Comparação entre o modelo OSI e o Fieldbus.



Fonte: (EMERSON, 2017)

2.2 CANbus

Fisicamente, a rede CAN (*Controller Area Network*) consiste em um barramento de transmissão digital que opera com vazão de dados entre 20 Kbps e 1 Mbit/s (NATALE et al., 2012), através de um protocolo de transmissão de dados de baixo custo e simples implementação. O sistema CAN (definido pelos padrões (ISO, 2003), (ISO, 2016), (ISO, 2006), (ISO, 2004), (ISO, 2007) e (ISO, 2013a)) é adequado para aplicações que requerem um grande número de mensagens curtas, que devem ser transmitidas em um ambiente eletromagneticamente hostil, sem comprometer a confiabilidade e eficiência da comunicação (JOHANSSON; TÖRNGREN; NIELSEN, 2005). Um quadro CAN 2.0A possui identificador de 11 bits, o que possibilita identificar 2048 (2^{11}) mensagens diferentes. Já no padrão *Extended* CAN (CAN Estendido), o quadro CAN 2.0B possui um identificador de 29 bits, viabilizando a identificação de 537 milhões mensagens diferentes (JOHANSSON; TÖRNGREN; NIELSEN, 2005).

Em aplicações veiculares, o padrão SAE J1939 é o mais utilizado e define as melhores práticas para implementar uma rede CAN. É tipicamente utilizado em veículos onde diversos nós (também conhecidos como unidades centrais eletrônicas — ECUs) são conectados à rede CAN. O padrão SAE J1939 define as cinco camadas superiores do modelo OSI e usa a CAN para as camadas física ((ISO, 2016) e (ISO, 2006)) e de enlace (ISO, 2003) (JUNGER, 2010). Numa rede CAN, o SAE J1939 recomenda como e quais dados são transmitidos entre as ECUs que controlam funções como, por exemplo, o motor, freios e transmissão, dentre outras. O padrão SAE J1939 utiliza um protocolo de comunicação *peer-to-peer* (cada um dos nós da rede funciona tanto como cliente quanto como servidor) por broadcast e gerenciamento de rede, através da arbitragem de mensagens.

2.2.1 Limitações do CANbus

A rede CAN tem sido por muitos anos a rede *backbone* da maioria dos veículos comercializados, por oferecer segurança e robustez na operação dos componentes eletrônicos do veículo. Todavia, o avanço tecnológico no meio automotivo trouxe consigo uma crescente busca por redes automotivas que oferecem uma vazão de dados que suporte simultaneamente todas as funções tradicionais do veículo, tais como carroceria, propulsor, dinâmica do veículo e multimídia. A rede CAN atende todas essas funções com eficiência, com exceção das novas funções multimídias, que demandam taxas de transmissão acima de 20 Mbit/s, enquanto que a rede CAN oferece taxas de transmissão limitadas a 1 Mbit/s, ou 6 a 8 Mbit/s no caso da CAN-FD (*CAN Flexible Datarate*).

Portanto, esta tradicional rede intraveicular não oferece desempenho suficiente para aplicações que demandem uma maior largura de banda (entre 10 Mbit/s e 100 Mbit/s), especialmente em aplicações que utilizem *streaming* de áudio e vídeo. Assim,

para viabilizar a incorporação de funções avançadas como multimídia, telemetria, IHM e ADAS, existe a necessidade de utilizar outras tecnologias de rede complementando a rede CAN, que atendam ao agregado da demanda. Existem algumas opções de novas tecnologias para rede intraveicular, algumas proprietárias como a *Media Oriented System Transport* (COOPERATION, 2004) que oferece taxas de transmissão da ordem de 25 a 150 Mbit/s utilizando de topologia em anel ((THIEL; KÖNIG, 1998); (TUOHY et al., 2015)), bem como tecnologias não proprietárias como a FlexRay (ISO, 2013b) que oferece taxas de transmissão de 2,5 Mbit/s a 10 Mbit/s utilizando de topologia barramento ("*party-line*") ou topologia estrela (CONSORTIUM et al., 2005).

Porém existem tecnologias não baseadas em sistemas *fieldbus* (TOVAR; VASQUES, 1999), ou seja, os dispositivos devem ter seu próprio conjunto de fios (cabeamento exclusivo) dedicado, em oposição à tradicional abordagem de barramento ou anel. Atualmente, dentre as tecnologias não proprietárias para redes intraveiculares, a Ethernet (IEEE 802.3bw, 2015) é considerada por órgãos de padronização internacionais e por importantes empresas do ramo automotivo como potencial substituta da rede CAN em aplicações veiculares ((IEEE 802.3bp, 2016); (VECTOR, 2016)), não somente pela maior taxa de dados oferecida (de 100 a 1000 vezes maior que o barramento CAN), mas também por sua notável e histórica capacidade de adaptação em diversos segmentos, como em redes industriais, comerciais e domésticas.

2.2.2 Quadros CAN

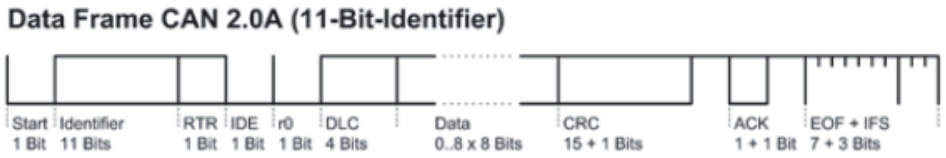
As mensagens trocados em um barramento CAN são chamadas de quadro e existem quatro tipos diferentes de quadros no protocolo CAN (FALLIS, 2013) sendo eles:

- *Data Frame* (Quadro de Dados)
- *Error Frame* (Quadro de Erro)
- *Remote Frame* (Quadro de Solicitação)
- *Overload Frame* (Quadro de Sobrecarga)

Desses quatro tipos de mensagens, apenas o *Data Frame* transporta mensagens, o restante é utilizado para contenção de erros, engatilhamento e sincronização.

Cada quadro possui uma sequência de bits (ID) que identifica o tipo de mensagem e sua prioridade. Existem dois tipos de identificadores diferentes, os de tamanho de 11 bits (especificado pelo CAN 2.0A) e de tamanho de 29 bits (especificado pelo CAN 2.0B) (OTHMAN et al., 2006). A estrutura de quadros CAN utilizada durante os testes e implementações neste trabalho correspondem as mensagens CAN 2.0A, com identificador de 11bits. A figura 3 mostra com detalhes a estrutura deste quadro.

Figura 3 – Estrutura do Quadro CAN 2.0A



Fonte: (FALLIS, 2013)

As funções e descrições de cada parte do quadro CAN 2.0A são as seguintes (FALLIS, 2013):

- Início do Quadro (*Start* - 1 bit): utilizado para marcar o começo do quadro;
- *Abritration Field* (*Identifier* + RTR - 12 bits): Possui 12 bits, sendo os 11 primeiros o identificador prioridade do quadro CAN e o último o RTR bit (*Remote-Transmission-Request* bit);
- *Control Field* (IDE + r0 + DLC - 6 bits): O primeiro bit dessa sequência de bits é o *Identifier Extension Flag* bit (IDE bit), que tem a característica de ser *low* (zero) e indica que o identificador está completo. O bit seguinte é o r0 (reservado). Os últimos 4 bits contém o *Data Length Code* (DLC) para o próximo campo, que é o campo de dados;
- *Data Field* (0-64 bits): conteúdo (*payload*) da mensagem;
- CRC *Field* (16 bits): contém o *checksum* para os bits anteriores. A sequência CRC (CRC *checksum*) é usada apenas para detecção de erros (não para a correção). O último bit representa o limitador do CRC, sendo o mesmo *high* (recessivo);
- *Acknowledge Field* (ACK - 2 bits): Todos os nós da rede que recebem uma mensagem CAN correta enviam um bit dominante (*low*) no ACK *slot* (o primeiro bit desse campo). Caso haja a detecção de um bit recessivo (*high*) o transmissor considera um erro de *acknowledgement*. O reconhecimento do bit dominante, não significa necessariamente que a mensagem foi entregue a todos os destinatários e sim que foi entregue para pelo o menos um. O último bit desse campo é um bit delimitador recessivo;
- *End of Frame* (EOF - 7 bits): são 7 bits recessivos (*high*) que indicam o fim do quadro CAN;
- *Inter Frame Space* (IFS - 3 bits): possui 3 bits recessivos e separa o quadro atual do próximo. Esse período também é utilizado pelos nós CAN para transferir mensagens

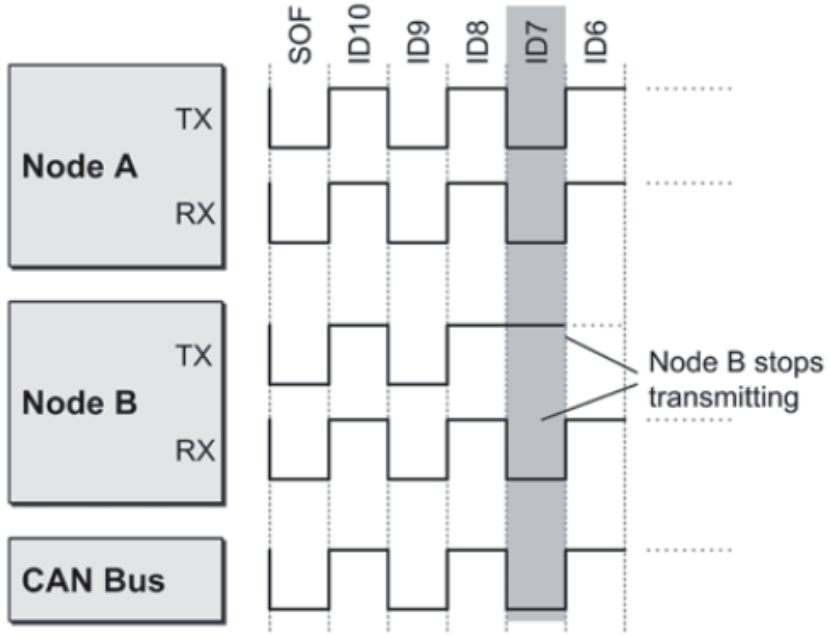
corretamente recebidas do controlador para o *buffer* de recebimento, ou do *buffer* de transmissão para o controlador.

2.2.3 Arbitragem de acesso ao barramento CAN

De acordo com o método de acesso ao meio CSMA/CD + AMP (*Carrier Sense Multiple Access/Collision Detection + Arbitration on Message Priority*) (CHEN; TIAN, 2009), dois ou mais nós podem iniciar sua transmissão ao mesmo tempo. Para que os nós possam transmitir suas mensagens, é necessário que o barramento esteja ocioso (*Carrier Sense*). Com o barramento ocioso, todos os nós que desejarem utilizar o mesmo, iniciam a transmissão enviando o *bit* dominante (*low*) de início de frame (*Multiple Access*). Cada nó CAN irá ler de volta o valor que está ocorrendo no barramento através do seu *transceiver* e irá comparar com o valor lógico transmitido (*Collision Detection*). Quando um nó com uma transmissão pendente percebe que o barramento está ocupado, ele espera até que o barramento esteja ocioso para tentar enviar sua mensagem. Quando isso acontece são reconhecidos *bits* recessivos no barramento. Os nós do barramento CAN podem enviar uma mensagem começando a transmissão pelo *bit* de início de quadro e sucessivamente o restante do mesmo desde que tenha a maior prioridade. A arbitragem para a utilização do barramento pelos nós CAN é feita *bit* a *bit* e de maneira não destrutiva. Isso quer dizer que a mensagem com maior prioridade, mesmo que enviada juntamente com outras mensagens de menor prioridade no barramento, não será corrompida durante sua transmissão inicial e não necessitará de retransmissão (FALLIS, 2013).

Na figura 4 temos um exemplo de transmissão e arbitragem no barramento CAN. Dois nós estão tentando se comunicar ao mesmo tempo no barramento. Os nós com menor identificador, possuem maior prioridade. É importante ressaltar que os *bits* dominantes são *low* (zero) e os *bits* recessivos são *high* (um). Como observamos na figura 4 a mensagem que esta sendo transmitida pelo nó A possui a seguinte sequência nos 4 primeiros *bits* do seu identificador: **1010** enquanto o nó B possui **1011**, logo o nó A possui prioridade maior que o nó B por ser menor. Os nós começam escrevendo juntos no barramento, inserindo primeiro o *Start of Frame* (que é um *bit* dominante) e em seguida os *bits* referentes ao identificador. Tanto o nó A quanto o nó B enviam um *bit* recessivo, um *bit* dominante e então um *bit* recessivo novamente (**101**). Até esse momento tanto o nó A quanto o nó B quando leem no barramento o mesmo valor de escrita, não detectando colisões. No quarto *bit*, o nó A escreve um *bit* dominante e o nó B escreve um *bit* recessivo, quando ocorre a leitura do sinal no barramento pelos nós A e B, o nó B percebe um *bit-error*, pois lê um valor diferente do valor que foi escrito por ele. Nesse momento o nó B reconhece que perdeu a arbitragem e então interrompe a transmissão ao barramento, enquanto o nó A por não perceber nenhuma diferença entre a leitura e a escrita prossegue com a transmissão do quadro.

Figura 4 – Exemplo de Transmissão e Arbitragem no Barramento CAN



Fonte: (FALLIS, 2013)

2.2.4 Gerenciamento de erros

Além dos *Bit-Errors* mostrados acima, existem outros erros que são tratados pelo CAN e as informações a seguir são referentes ao gerenciamento de erros que acontece na camada de enlace. O CAN tem o princípio de que a maior quantidade de erros possível deve ser tratada pelos nós CAN. Quando ocorre erro em algum quadro, todos os nós da rede são comunicados e então esse quadro é descartado. A correção desse erro é feita através da retransmissão (função da camada de enlace). De maneira geral o gerenciamento de falhas é feito em três etapas: detecção de erro, tratamento de erro e isolamento de falha. As características dessas etapas são descritas a seguir (FALLIS, 2013).

2.2.4.1 Detecção de Erros

Os seguintes erros podem ocorrer, sendo detectados e tratados em um barramento CAN:

- *Bit-Error*: Ocorre quando o bit lido tem valor lógico diferente do escrito;
- *Stuff-Error*: Mais de cinco bits do mesmo nível são detectados (exceto no EOF e no IFS);
- *CRC-Error*: Erro detectado pelo cálculo do *checksum*;

- *Form-Error*: Ocorre quando há violação no formato geral do quadro CAN;
- *Acknowledgement-Error*: Transmissor não detecta o bit dominante no *ACK Slot*, o que significa que a mensagem não foi reconhecida como uma mensagem válida por outro nó do barramento CAN.

2.2.4.2 Tratamento de Erros

Se um dos erros citados acima é detectado, os nós da rede CAN são notificados com o *error-frame*. O *error-frame* sobrescreve outros quadros e é detectado pelos nós através da violação das regras de *bit-stuffing*[1]. Quando ocorre um erro local no barramento CAN (erros que não são detectados por todos os nós ao mesmo tempo), os nós detectam a violação nas regras do *bit-stuffing* e então começam a enviar *error-frames*. Nesse momento a mensagem que está atualmente em progresso é rejeitada pelos nós e reenviada posteriormente (FALLIS, 2013).

2.2.4.3 Isolamento de Falhas

Com o objetivo de isolar falhas, os nós no barramento CAN possuem três estados para que seja possível o controle e o isolamento de nós específicos causando perturbação no barramento (FALLIS, 2013). Esses estados são ilustrados pela figura 5 e explicado com detalhes a seguir.

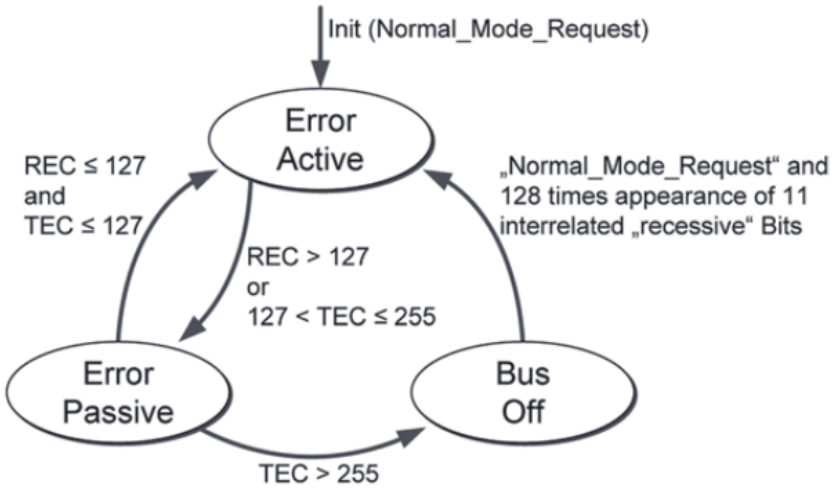
- *Error Active*: é o estado normal de um nó CAN. Neste estado o nó pode enviar e receber mensagens sem nenhum tipo de restrição. No caso de eventuais falhas, um sinal de *Active Error Flag* (sequências de *bits* dominantes) é transmitida ao barramento;
- *Error Passive*: é o estado que um nó assume após muitos erros consecutivos. Neste estado o nó CAN pode continuar enviando e recebendo mensagens, porém quando ocorre algum erro um sinal de *Passive Error Flag* (sequência de *bits* recessivos) é enviado, não interferindo na transmissão das outras mensagens no barramento;
- *Bus Off*: ocorre quando o nó é totalmente desconectado do barramento. O nó recebe este *status* quando causa perturbações longas ou frequentes no barramento e fica impedido de transmitir mensagens e/ou *flags* de erro.

2.2.5 Considerações Sobre a Latência

Em uma rede CAN sob circunstâncias normais, é possível calcular a latência mínima apenas para as mensagens de maior prioridade. O atraso para mensagens de menor

[1] https://en.wikipedia.org/wiki/Bit$_s tuffing$

Figura 5 – Estados dos Nós CAN e suas Transições



Fonte: (FALLIS, 2013)

prioridade não é determinístico, podendo apenas ser mensurado estatisticamente ou através de simuladores e analisadores de redes, uma vez que elas serão postergadas sempre que uma mensagem de maior ID tentar ser transmitida simultaneamente no barramento. Uma maneira de calcular um limite para o atraso no envio de uma mensagem de baixa prioridade é considerar o pior caso no seu tempo de resposta (*Worst Case Response Time* - WCRT), ou seja, quando todas as mensagens tentarem ser transmitidas ao mesmo tempo e a de menor prioridade perder a arbitragem para toda e cada uma das demais (TINDELL; BURNS; WELLINGS, 1995).

Quanto ao atraso máximo para acesso ao barramento, dentro de um quadro CAN, cada mensagem pode conter de 0 a 64 *bits* de dados (até 8 bytes). Considere que os dois tipos de quadros CAN (ID de 11 ou 29 *bits*) possuem diferentes latências mínimas e taxas de transmissão efetiva para mensagens de altas prioridades. Considerando uma rede CAN com taxa de transmissão de 1 Mbit/s que possui apenas quadros CAN com identificador de 11 *bits*, a mensagem com maior prioridade tem que esperar no máximo 135 *bits* (quantidade máxima de *bits* de uma mensagem que já está no barramento) para acessar o barramento, que resulta em um total de 135 μs. No caso de um barramento que possui quadros com identificadores de 29 *bits* o máximo de espera necessário é de 160 *bits*, que equivale a 160 μs na mesma taxa de transmissão de 1 Mbit/s.

2.3 Ethernet

A rede Ethernet veicular (ou Ethernet automotiva) é um enlace utilizado a fim de se conectar diversos componentes dentro do veículo usando uma rede local cabeada

(LAN). Os padrões Ethernet (IEEE 802.3, 2009) vem sendo adaptados a fim de atender às necessidades específicas do mercado automobilístico, aperfeiçoando o atendimento de requisitos tais como interferências eletromagnéticas e de radiofrequências, oferta da largura de banda, garantias de latência, sincronização e gerenciamento de rede. A flexibilidade e escalabilidade da tecnologia Ethernet irá melhorar dramaticamente a segurança do veículo, conforto e sistema de entretenimento, reduzindo significativamente os custos e complexidade da rede e cabeamento (ALLIANCE; BROADR-REACH, 2014). A proposta de adoção da tecnologia Ethernet como rede intraveicular acarreta redução no cabeamento (e seu consequente peso e custo), com eliminação gradual de redes complementares no veículo (ex.: CAN-FD, FlexRay, LVDS) e consequente redução de custos (SAUERWALD, 2014). Porém, deve-se ressaltar que a adaptação da tecnologia Ethernet para aplicações no setor automotivo enfrenta alguns desafios.

Para que a tecnologia se torne viável em termos econômicos, projetos de PD&I têm sido promovidos para possibilitar e aprimorar o funcionamento da Ethernet no ambiente veicular com a utilização de apenas um par de fios, diferentemente do que acontece na sua aplicação convencional em rede de computadores, onde se faz o uso de quatro pares de fios trançados. A partir do pioneiro padrão **BroadR-Reach** (HANK; SUERMANN; MÜLLER, 2012), a **OPEN Alliance** e **IEEE** desenvolveram novas tecnologias baseadas na tradicional rede 1000BASE-TX "Gigabit Ethernet" (IEEE 802.3ab, 2015). A partir dele, a Ethernet veicular passa a proporcionar transmissões de dados através de um único par de fios de cobre, em ambas direções (*full-duplex*), denominado 100BASE-T1 e padronizado pela norma (IEEE 802.3bw, 2015) cláusula 96. Esta especificação faz uso de avançados recursos de cancelamento de eco, modulação (PAM-3) e codificação (DSP *scrambler*). O 100BASE-T1 também possui requisito de comprimento máximo de 15 metros e propõe a alteração da frequência de trabalho de 125 MHz (Gigabit Ethernet) para 66 MHz (HOGENMÜLLER, 2014), para viabilizar que a rede Ethernet Veicular atenda aos requisitos do setor automotivo para interferências eletromagnéticas e de radiofrequência (KERN, 2012).

Apesar da grande capacidade de transmissão de informação útil por mensagem, um pacote Ethernet também possui sobrecarga de protocolo (*protocol overhead*)[2], que consiste na quantidade de bytes extras utilizados pelo protocolo (como por exemplo cabeçalho e cauda) para o envio da mensagem efetiva, que pode mensurada em quantidade de bytes adicionais à mensagem ou em forma de taxa, que representa a relação entre o conteúdo da mensagem e os bytes extras. Além do tamanho dos quadros de carga útil, a rede Ethernet difere drasticamente da rede CAN no que diz respeito à topologia de rede utilizada. A rede CAN utiliza de um *fieldbus* clássico, sendo implementada sob uma topologia de barramento, no qual uma mensagem é transmitida a todos os nós e cada

[2] https://en.wikipedia.org/wiki/Overhead$_{(}computing)Communications_{.}28data_{t}ransfer_{o}verhead.29$

nó decide se essa mensagem é relevante a ele e se será posteriormente processada. Numa rede CAN, a arbitragem para acesso ao barramento (caso mais de um nó tente fazê-lo simultaneamente) é feita por meio do campo ID no cabeçalho do quadro, que define a prioridade em ordem crescente de ID. Também, não existe a opção de enviar a mensagem a um único destinatário. Esse tipo de envio de mensagem a todos os nós caracteriza a rede CAN como um meio *broadcast*. Diferente das redes veiculares *fieldbus*, a rede Ethernet faz uso de uma topologia em árvore ("ramificada"), tal como apresentado na Figura 1, fazendo uso de comutadores (*switches*) que limitam o domínio de colisões de mensagens e oferece entrega ponto-a-ponto das mensagens.

Tal qual o ID da rede CAN, cada nó da rede Ethernet tem o seu próprio endereço físico MAC (*Media Access Control*), o qual é usado como endereço único do dispositivo na rede, ou seja, o nó só processa a mensagem se a mensagem tiver ele como destinatário específico ou se for uma mensagem broadcast, direcionada para um endereço especial (FF:FF:FF:FF). Essa relação de comunicação 1:1 (um-para-um) é chamada de endereçamento *unicast*, na qual o comutador somente envia o quadro ao barramento onde está localizado o nó destinatário. Além disso, o *switch* permite o envio de quadros *multicast*, direcionados a determinado grupo de nós interessados. Em suma, o Ethernet provê troca de mensagens de 1:1 (*unicast*), 1:n (*multicast*) e 1:*all* (*broadcast*).

Entretanto o Ethernet tradicional não garante latência ultrabaixa, necessária numa rede intraveicular para a comunicação entre sensores e atuadores, que exigem um tempo de reação rápido(ex.: menor que 10 μs nos sistemas de propulsão e dinâmica). A rede Ethernet Veicular busca sanar esta deficiência ao adotar e aprimorar o padrão AVB para redes com restrições temporais. Também, tradicionalmente a Ethernet não tem um modo de controle da alocação da largura de banda para diferentes aplicações, ao passo que sua aplicação veicular adotou o padrão AVB e relacionados para reserva de banda e alocação de slots de tempo para transmissão. A tendência é de que em curto ou médio prazo estes desafios tenham sido superados com êxito e a rede Ethernet se torna a única rede Veicular utilizada em sistemas veiculares, robusta o suficiente para passar pelos requisitos exigidos pelo setor automotivo.

2.4 Redes Sensíveis à Temporização

Redes **TSN**[3] (*Time Sensitive Networks*) são um conjunto de padrões em desenvolvimento por um grupo de trabalho da IEEE 802.1 desde 2012, definindo mecanismos de transmissão de dados sensíveis a temporização, através de extensões ao padrão de redes virtuais (VLANs) IEEE 802.1Q[4]. Redes TSN visam prover comunicação confiável sobre Ethernet, provendo latência ultrabaixa (*delay* $\leq$ 10 μs), alta disponibilidade e determi-

[3] http://www.ieee802.org/1/pages/tsn.html
[4] http://www.ieee802.org/1/pages/802.1Q.html

nismo nas latência fim-a-fim. A utilização da rede Ethernet para aplicações multimídia foi padronizada em 2011 pelos **protocolos AVB** ((IEEE 802.1BA, 2014) (IEEE 802.1AS, 2013) (IEEE 802.1Qat, 2010) (IEEE 802.Qav, 2009)). A fim de melhorar o comportamento determinístico e a transferência de dados assíncrona apresentada pela rede Ethernet, em 2009 diversas montadoras formaram o consórcio *AVnu Alliance*[5], para desenvolver e aprimorar o protocolo AVB, adicionando melhorias na temporização. Os novos protocolos AVB definem mecanismos mais precisos de sincronização na transferência de dados com uma garantia de limite para a latência.

Numa rede **Ethernet AVB**, é reservada uma fração da largura de banda disponível para tráfego de áudio e vídeo (AVB). Os pacotes AVB são transmitidos regularmente em slots de tempo específicos (TDMA). Como controle de acesso ao meio passa a ser dividido e organizado, não haverá colisões ao contrário do CSMA/CD. Para tal, todos os nós do sistema compartilham de um *clock* virtual. Todos os pacotes AVB tem uma divisão temporal que garante a transmissão dos dados sem colisões e consequentes perdas de dados. Protocolos automotivos proprietários já empregados no meio automotivo, tais como **MOST** (THIEL; KÖNIG, 1998), já fazem uso do AVB, porém, no caso do MOST a largura de banda total é compartilhada entre todos os nós (sem reserva), o que resulta em eventuais colisões e perda de pacotes. Do ponto de vista de construção física, o protocolo Ethernet AVB proporciona uma redução significativa na fiação requerida para aplicações de áudio e vídeo de tempo real (AVB), apresentando benefícios como redução do peso, custos e consumo de combustível no mercado automobilístico (GOTHARD et al., 2014).

Um outro exemplo de rede TSN é a tecnologia **AFDX**[6] (*Avionics Full Duplex Ethernet*), padrão de comunicação em aeronaves modernas que utilizando topologia em estrela para reduzir o comprimento e peso do cabeamento (YUNA; XIONG, 2008). Uma importante característica do AFDX é o determinismo da rede: cada nó tem acesso livre à rede, sendo estabelecidos enlaces virtuais entre um transmissor e vários receptores no qual o tempo para transmissão dos dados é limitado e computado por uma abordagem formal de *Network Calculus* (SCHNEELE, 2015). Já o **EtherCAT**[7] (*Ethernet for Control Automation Technology*) é uma sistema *fieldbus* baseado em Ethernet, padronizado no IEC 61158 e apropriado para uso em aplicações de automação com requisitos de tempo-real, que necessitem de curtos intervalos de atualização (*cycle times* ≤ 100 μs) com baixa variação na latência de comunicação (*jitter* ≤ 1 μs) para sincronização precisa (BUTTNER, 2004). Ainda, existem tecnologias de rede com gatilhos de tempo, como o protocolo **TTP**[8] (*Time-Triggered Protocol*) que consiste em um barramento de canal duplo com 25 Mbit/s cada (KOPETZ; GRUNSTEIDL, 1993).

[5] http://avnu.org
[6] https://en.wikipedia.org/wiki/Avionics$_{F}ull - Duplex_{S}witched_{E}thernet$
[7] https://en.wikipedia.org/wiki/EtherCAT
[8] https://en.wikipedia.org/wiki/Time-Triggered$_{P}rotocol$

2.5 Trabalhos Relacionados

Existem diversos trabalhos que abordam temas relacionados à configuração de pontes, estratégias para a multiplexação e enfileiramento de quadros CAN em mensagens Ethernet e vice-versa. Esta seção apresenta algumas estratégias, bem como ressalta quais foram quais foram as variações e novas ideias implementadas neste trabalho.

Em (KERN et al., 2011) são apresentadas um total de quatro estratégias e variações para a criação de pontes de quadros CAN em pacotes Ethernet e vice versa. Essas estratégias são as seguintes:

2.5.1 Estratégias de Encapsulamento e Enfileiramento

2.5.1.1 Variante 0: Design de Referência

A estratégia mais básica e intuitiva é a Variante 0, que consiste em um mapeamento de Um-para-um dos quadros CAN em mensagens Ethernet. Dessa maneira cada quadro CAN tem o *payload* de uma mensagem Ethernet dedicado a ele, o que causa um alto custo para o encapsulamento (*protocol overhead*), porém apresenta baixa latência adicional no envio das mensagens.

2.5.1.2 Variante 1: Abordagem de Buffer

Esta estratégia consiste na utilização de um *buffer* para o armazenamento e envio de vários quadros CAN dentro de uma mensagem Ethernet, diminuindo assim o *overhead* causado pelos protocolos de alto nível (tais como o Ethernet e o IP). O envio da mensagem contendo o *buffer* pode ser ativado através de duas variações dentro desta estratégia apresentada em (KERN et al., 2011):

- *Buffer* Cheio: É determinado um tamanho para o *buffer* (de acordo com a capacidade do payload da mensagem Ethernet) e assim que esse *buffer* é preenchido por quadros CAN a mensagem é enviada;
- *Timeout*: É definido um tempo máximo para o *buffer* continuar recebendo quadros CAN. Se esse tempo for atingido a mensagem Ethernet é enviada com o *buffer*, estando este cheio ou não.

2.5.1.3 Variante 2: Abordagem Temporizada

A abordagem temporizada leva em consideração que o protocolo CAN utiliza um conceito de prioridade para o escalonamento das mensagens que são compartilhadas no barramento. A principal ideia desta estratégia é colocar um temporizador no *buffer* de quadros CAN e, de acordo com a prioridade dos quadros que vão sendo alocados neste *buffer*,

diminuir o temporizador até que chegue a zero e então despachar a mensagem Ethernet com o *buffer* em seu *payload*. Esta abordagem faz com que *buffers* que contenham quadros com alta prioridade sejam enviados mais rapidamente, simulando o comportamento do barramento CAN.

2.5.1.4 Variante 3: Abordagem de Urgência

Similar a variante 2, a abordagem de urgência tem como objetivo diminuir a sobrecarga causada pelo cabeçalho Ethernet (*protocol overhead*) e ao mesmo tempo simular o comportamento do barramento CAN no momento do enfileiramento. A abordagem de urgência preenche o *buffer* de quadros CAN até que uma mensagem de alta prioridade surja no barramento. Quando isso acontece, essa mensagem de alta prioridade é colocada dentro do buffer, que imediatamente é despachado no pacote Ethernet. A aplicação desta estratégia faz com que os quadros CAN de alta prioridade sofram baixíssima latência no momento de enfileiramento na ponte, porém, pode não aproveitar a vazão máxima proporcionada pelo enlace Ethernet.

2.5.1.5 Múltiplas Filas

Além das estratégias exemplificadas anteriormente, é apresentado em (LEE; PARK, 2013) uma abordagem de redução da Unidade Máxima de Transmissão (MTU), que consiste na redução do tamanho do maior pacote que o protocolo pode transmitir e possui como consequência o aumento da sobrecarga de protocolo e diminuição da vazão. Neste artigo os autores utilizam uma estratégia de duas filas, uma para as quadros com prioridade maior e outra para quadros com menor prioridade, fazendo com que a redução na vazão causada pela redução da Unidade Máxima de Transmissão seja compensada por mais filas. Neste trabalho não foram utilizadas múltiplas filas devido ao foco de experimentação nas variações de estratégias de enfileiramento e multiplexação em uma única fila (*buffer*).

2.5.2 Adaptação de Ciclo e Comutação de Pacotes Ethernet

Em (NACER et al., 2013) é apresentada a visão geral de uma ponte que realiza a intercomunicação entre barramentos CAN, mostrando a utilização de pontes de entrada e saída para quadros CAN e, entre eles, um *switch* que direciona os pacotes Ethernet para a interface (barramento) correta de destino dos quadros. Um diferencial na implementação das estratégias desse trabalho é que, além de utilizarem as abordagens de um quadro CAN para um quadro Ethernet (um-para-um) e vários quadro CAN para um quadro Ethernet (n-para-um), os autores realizaram dois experimentos: o envio imediato dos quadros CAN recebidos e a aplicação proposital de um atraso ao escrever os quadros no barramento CAN para ajustar os ciclos.

2.5.3 Abordagem do Trabalho

Em relação às diversas técnicas e estratégias apresentadas acima, o foco do trabalho foi na implementação, adaptação e criação de novas técnicas de enfileiramento e multiplexação de quadros CAN em pacotes Ethernet utilizando uma única fila. Através da implementação, teste e análise de algumas das abordagens acima descritas, novas técnicas foram implementadas pelo autor. Em destaque, foram propostas e implementadas abordagens baseadas no ciclo da mensagem CAN ao invés do seu ID, inspiradas em trabalhos como o de (TINDELL; BURNS; WELLINGS, 1995). Este trabalho buscou explorar outras variáveis que influenciam no comportamento de uma ponte entre protocolos CAN e Ethernet, além do tradicional CAN ID, assim como estudar qual o comportamento dessas estratégias e quais cenários seriam apropriados para cada comportamento.

3 METODOLOGIA

3.1 Materiais

Para o projeto e implementação deste trabalho, foram utilizados os seguintes *softwares*: a linguagem C (ISO, 1999), o compilador GCC[1] e editor de texto Atom[2]; ferramentas de captura, exibição e geração de pacotes (`can-utils`[3] e SocketCAN[4]), softwares para análise estatística (R-*Project*[5], *Apache Commons Math*[6]) e BASH *scripts* (vide GitHub[7]).

3.1.1 Bibliotecas

As distribuições Linux possuem suporte ao CAN graças a um conjunto de ferramentas chamado SocketCAN que foi desenvolvido pela Volkswagen em contribuição ao *Kernel* Linux. Já a biblioteca `can-utils`[8] é um conjunto de ferramentas do próprio SocketCAN que fornece diversos *scripts* para gerar, capturar, exibir e reproduzir tráfego CAN. Esta biblioteca possibilita a simulação de interfaces virtuais CAN e, juntamente com os scripts para a manipulação de tráfego CAN proporciona o ambiente necessário para o desenvolvimento de algoritmos e protótipos antes de serem efetivamente testados em ambientes automobilísticos reais. Como o `can-utils` é um subconjunto de ferramentas do próprio SocketCAN, está disponível para distribuições Linux, seja para o Raspberry Pi ou para computadores de mesa e notebooks.

3.1.2 Equipamentos

A seguir será apresentada a lista dos equipamentos utilizados na realização deste trabalho, assim como suas respectivas descrições.

3.1.2.1 Raspberry Pi

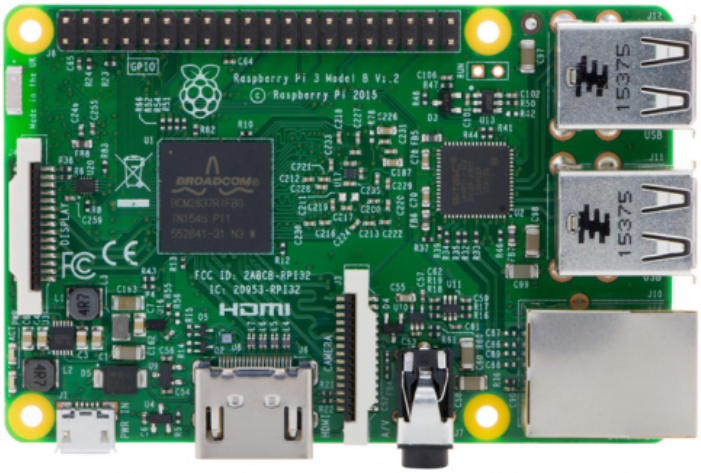
O Raspberry Pi[9] foi utilizado neste trabalho como plataforma de desenvolvimento, sendo o equipamento responsável por realizar o enfileiramento das mensagens CAN, mul-

[1] https://gcc.gnu.org/onlinedocs/gcc/Standards.htmlStandards
[2] https://atom.io/
[3] https://github.com/linux-can/can-utils
[4] https://en.wikipedia.org/wiki/SocketCAN
[5] https://www.r-project.org/
[6] https://commons.apache.org/proper/commons-math/userguide/stat.html
[7] https://github.com/ribeirorenan/tcc-multiplexacao-can-ethernet
[8] https://github.com/linux-can/can-utils/blob/master/README.md
[9] https://www.raspberrypi.org/

tiplexação dentro de um quadro Ethernet e seu envio, bem como atuou como um *gateway* para os enlaces Ethernet aos quais foram conectadas as pontes CAN-Ethernet. O Raspberry Pi consiste em um computador do tamanho de um cartão de crédito, permitindo a utilização de um sistema operacional que seja compatível com a arquitetura ARM, sendo que utilizamos o Raspbian [10] *Release Date 2017-07-05 jessie-lite (Minimal Version)*. O modelo adotado foi o **Raspberry Pi 3 Model B**[11](Figura 6), cuja a especificação é:

- Processador *Quad Core* 1,2 GHz (0,8 *ns*) Broadcom BCM2837 64-bits;
- Memória RAM de 1 GB;
- 40 pinos de GPIO estendidos;
- 4 entradas USB 2.0 e saída HDMI padrão;
- Entrada microSD para persistência de dados e carregamento do S.O;
- Entrada microUSB para fonte de energia de até 5 V e 2,5 A.

Figura 6 – Raspberry Pi 3 Model B



Fonte:(FOUNDATION, 2017)

3.1.2.2 Arduino Mega 2560

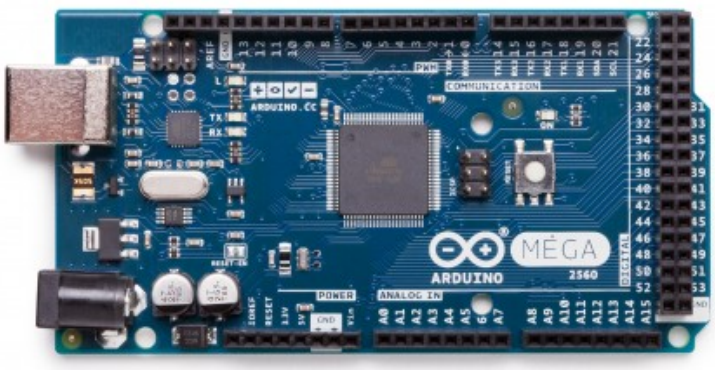
O **Arduino Mega 2560 R3**[12] (Figura 7) foi utilizado para controle de transceptores CAN e testes de compatibilidade de transceptores e seus níveis de tensão. Esta versão do Arduino possui a seguinte especificação:

- Microcontrolador ATmega2560 com clock de 16 MHz (62,5 *ns*)

[10] https://www.raspbian.org/
[11] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[12] https://store.arduino.cc/usa/arduino-mega-2560-rev3

- 256 KB de memória Flash (8KB para o *bootloader*), 8 KB SRAM, 4 KB EEPROM;
- Entrada/saída: 54 digitais (15 PWM) e 16 entradas analógicas, comunicação SPI, UART, I2C;
- Tensão de operação: 5 V / Corrente DC de operação: 20-50 mA;
- Conexão serial TTL, programável com a utilização de conversor FTDI USB/Serial;
- Conexão USB e conexão alternativa para alimentação externa;
- Tensão de entrada: 7–12 V.

Figura 7 – Arduino Mega 2560 R3



Fonte: (ARDUINO, 2017)

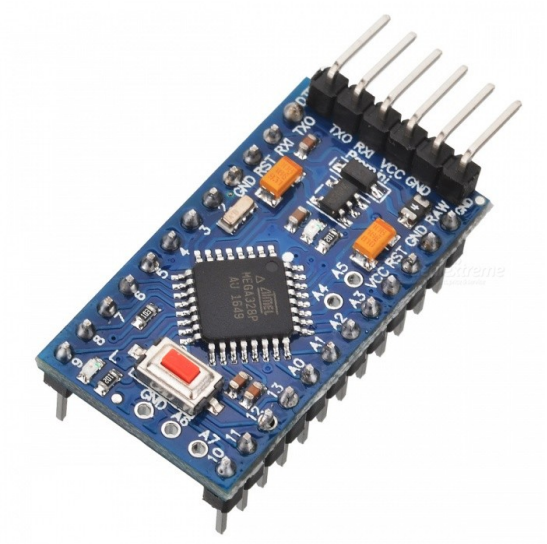
3.1.2.3 Arduino Pro Mini

O **Arduino Pro Mini**[13] (Figura 8) foi utilizado para o controle de transceptores CAN, tendo a seguinte especificação:

- Microcontrolador ATmega328p com *clock* de 16 MHz (62,5 *n*s)
- 32KB de memória Flash (2KB pro *bootloader*), 1 KB SRAM, 1 KB EEPROM;
- Entrada/saída: 14 digitais (6 PWM) e 8 entradas analógicas, comunicação SPI, UART, I2C;
- Tensão de operação: 3,3 – 5 V / Corrente DC de operação: 40 mA;
- Conexão serial TTL, programável com a utilização de conversor FTDI USB/Serial;
- Tensão de entrada: 3,35 – 12 V.

[13] https://store.arduino.cc/usa/arduino-pro-mini

Figura 8 – Arduino Pro Mini

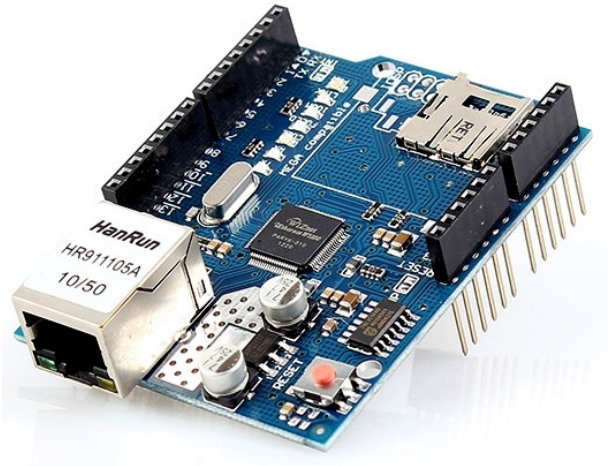


Fonte: (DX, 2017a)

3.1.2.4 Shield Ethernet W5100

O **Shield Ethernet W5100** foi utilizado em conjunto com o Arduino Uno e Mega para que fosse possível a comunicação dos microcontroladores com outros dispositivos que possuem interface Ethernet (por exemplo o Raspberry Pi). A Figura 9 mostra como é este *shield*.

Figura 9 – Shield Wiznet 5100



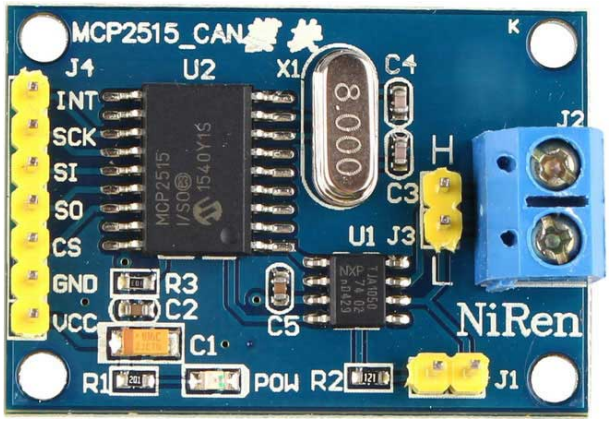
Fonte: (FILIPEFLOP, 2017)

3.1.2.5 Módulos CAN

Foram utilizados **Módulos CAN bus** (Figura 10) da marca NiRen, que utilizam:

- Controlador MCP2515: controlador CAN *stand-alone* com 18 pinos e interface SPI;
- Transceptor TJA1050: interface entre o controlador CAN e o barramento físico.

Figura 10 – Módulo Can MCP2515 Niren



Fonte: (DX, 2017b)

3.1.2.6 Conversor de Nível Lógico

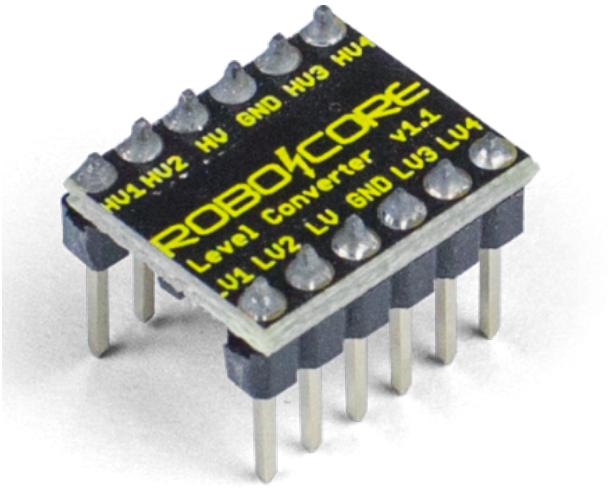
O Conversor de Nível Lógico Robo Core[14] (Figura 11) foi utilizado para possibilitar a comunicação entre os módulos CAN bus e o Raspberry Pi. Os pinos de entrada e saída de propósito geral do Raspberry Pi utilizam um nível de tensão de 3,3 V, enquanto o módulo CAN necessita de 5 V para o melhor funcionamento do mesmo. A função do conversor de nível lógico é realizar as conversões de 5 V para 3,3 V quando o Raspberry recebe um sinal do módulo e converter de 3,3 V para 5 V quando é enviado um sinal do Raspberry para o conversor.

3.1.3 Simulação

Através das ferramentas providas pela biblioteca `can-utils` e SocketCAN, foi possível realizar a simulação de interfaces virtuais CAN dentro do próprio sistema operacional do Raspberry Pi. Essas interfaces virtuais são chamadas de **vcan**, sendo possível a criação de múltiplas interfaces virtuais. O `can-utils` também disponibiliza *scripts* que permitem a simulação de um barramento CAN e essas ferramentas foram utilizadas durante a fase de desenvolvimento dos algoritmos de multiplexação de pacote. As principais ferramentas utilizadas foram as seguintes:

[14] https://www.robocore.net/loja/produtos/conversor-de-nivel-logico.html

Figura 11 – Conversor de Nível Lógico Robo Core



Fonte: (PETREVSKI, 2017)

- **`cansend`**: Envia uma mensagem CAN específica para alguma interface, tanto virtual quanto física;
- **`cangen`**: Gera tráfego na interface CAN especificada, podendo ser tanto virtual quanto física. O tráfego gerado pode ser aleatório ou com mensagens predefinidas;
- **`candump`**: Monitora uma interface CAN, mostrando todas as mensagens que passam pelo barramento em que aquela interface se encontra e funciona com interfaces físicas e virtuais. Possui a opção de gerar um arquivo de *log* com o tráfego obtido;
- **`canplay`**: Reproduz o tráfego CAN que foi salvo em arquivo de *log*. O formato de *log* que é reproduzido pelo `canplay` é o mesmo gerado pelo `candump`, possibilitando a reprodução de um mesmo *log* obtido pelo `candump` inúmeras vezes (repetindo-o);
- **`cansniffer`**: Monitora uma interface CAN, porém diferentemente do `candump` o `cansniffer` realiza uma análise dos quadros que estão sendo monitorados e mostra aspectos como ciclos e IDs de maior prioridade;
- **`canbusload`**: Mostra a porcentagem de carga do barramento através de uma interface CAN.

3.2 Métodos

No primeiro momento foram conduzidos estudos bibliográficos para compreensão da arquitetura *fieldbus* e da rede CAN bus. Também, foi realizado um levantamento do estado da arte em relação ao desenvolvimento da tecnologia Ethernet veicular e protocolos afins, especialmente no que tange a temporização e propostas de mecanismos de interconexão dessas duas redes num ambiente veicular.

Em seguida, foram comparadas estratégias para incorporação de pacotes CAN den-

tro de quadros Ethernet, a fim de identificar técnicas promissoras de multiplexação que visem garantir os princípios de temporização e prioridade previstos pelo protocolo CAN. Tipicamente, as estratégias abordam aspectos tais como escalonamento na fila, uso de múltiplas filas, enfileiramento com prioridade e/ou uso de gatilhos de tempo, dentre outros. As estratégias de multiplexação avaliadas e as novas propostas foram implementadas na linguagem de programação C. No projeto e implementação das estratégias de multiplexação, visamos garantir a priorização e/ou temporização esperadas pelas mensagens CAN, bem como uma boa utilização do enlace Ethernet ao tentar minimizar a sobrecarga de protocolo (*protocol overhead*, conforme visto na seção 2.3).

Para analisar o desempenho dos algoritmos propostos e validar a prova de conceito, foram projetados e configurados experimentos de simulação e testes de bancada. Conforme descrito nas próximas seções, foram implementados protótipos de ponte CAN-Ethernet, através de microcontroladores e transceptores (CAN e Ethernet). Através da análise dos resultados, coletados nos experimentos de teste e validação, os algoritmos com a estratégia de enfileiramento e multiplexação foram ajustados, modificados e calibrados para melhor atender às restrições de temporização e prioridade previstas pelo CANbus.

3.2.1 Projeto

3.2.1.1 Encapsulamento de Mensagens

Um pacote Ethernet possui um espaço reservado para envio de informações (*payload*) de no mínimo 46 bytes e no máximo 1500 bytes (IEEE 802.3, 2015). Quando são utilizadas as bibliotecas nativas do Linux para a manipulação de *socket* Ethernet, basta especificar o endereço da estrutura (ponteiro pro *buffer*) utilizada para armazenar os dados do *payload* no momento de envio da mensagem.

Para o encapsulamento de apenas um quadro CAN dentro de um pacote Ethernet, o endereço da própria estrutura do quadro CAN é passada para a função de envio como *payload*, que é interpretada como um arranjo de bytes. Quando o destino recebe o pacote Ethernet, é necessário ler apenas os bytes referentes à estrutura do quadro CAN e então realizar uma conversão para que as informações daquele quadro possam ser acessadas novamente no seu formato original. Para o encapsulamento de mais de um quadro CAN em um mesmo pacote Ethernet (processo conhecido como multiplexação), é necessário que conste no *payload* Ethernet a quantidade total de quadros CAN que ele transporta, para fins de demultiplexação. Como foi utilizado o SocketCAN, os quadros CAN gerados na simulação eram armazenados em uma estrutura de dados (*struct can_frame*) de tamanho fixo de 16 bytes, proveniente da própria biblioteca. Dessa maneira foi suficiente armazenar apenas a quantidade de quadros CAN no início do *payload* Ethernet e utilizar aritmética de ponteiros para recuperar os quadros armazenados. Este tópico é abordado em detalhes na seção de Multiplexação de Mensagens.

3.2.1.2 Multiplexação de Mensagens

Um dos objetivos de se utilizar a tecnologia Ethernet no meio automobilístico é aumentar a vazão da quantidade de informações que é possível transmitir, sendo portanto desejado obter o melhor aproveitamento da capacidade fornecida pelo Ethernet. Para isso, é necessário preencher com a maior quantidade possível de mensagens CAN o *payload* das mensagens Ethernet para que a sobrecarga de protocolo (bytes adicionais à mensagem, tal como cabeçalho e cauda, conforme tratado na seção 2.3) cause menor impacto na vazão final. Neste trabalho, a estrutura utilizada para armazenar um quadro CAN ocupa 16 bytes na memória, portanto é possível inserir no máximo 93 quadros CAN no *payload* Ethernet com seu tamanho máximo de 1500 bytes e, como o *payload* Ethernet tem obrigatoriamente no mínimo 46 bytes (IEEE 802.3, 2015), é absolutamente um desperdício enviar uma quantidade menor do que 3 quadros CAN por pacote Ethernet.

Quando o número de quadros a serem enviados é fixo, basta definir uma constante (com a quantidade de quadros armazenados em cada mensagem Ethernet) nos dispositivos que vão enviar e receber as mensagens Ethernet. No momento de envio deve-se armazenar os quadros CAN de 16 em 16 bytes em uma estrutura que possa ser enviada como *payload* (por exemplo um vetor de bytes) e no momento do recebimento o *payload* deve ser lido também de 16 em 16 bytes de acordo com a constante definida, realizando a conversão necessária para a estrutura original do quadro. Porém, se a quantidade de quadros CAN a serem enviados não é fixa, é armazenado no primeiro byte do *payload* o número de mensagens que vão ser enviadas, dessas maneira quando um pacote Ethernet é recebido, o primeiro byte é convertido em inteiro e então lê-se de 16 em 16 bytes o restante do *payload* de acordo com o número de mensagens contidas.

Realizando a multiplexação de quadros CAN da maneira mais simples possível, apenas escrevendo os bytes no *payload*, faz com que o *overhead* causado pelo processamento e pela transmissão de dados seja menor, diminuindo consequentemente a latência final gerada pelos algoritmos de enfileiramento.

3.2.1.3 Comunicação Entre Redes

Esta seção mostra de maneira sucinta como foi desenvolvida e implementada a estrutura básica de comunicação entre os protocolos CAN e Ethernet, que em suma consiste na implementação da estratégia de multiplexação denominada variante 0 em (KERN et al., 2011) mas referenciado como Um-para-um neste trabalho de conclusão de curso. A ideia desta abordagem é o envio de um quadro CAN em um pacote Ethernet dedicado. A partir desta implementação foi possível dispor de uma estrutura básica para a aplicação das outras ideias de multiplexação abordadas neste trabalho.

Inicialmente foram criadas duas interfaces virtuais (disponíveis pelo `can-utils`),

uma chamada de **vcan0** e outra chamada **vcan1**. Para realizar a comunicação entre essas duas interfaces, um lado da ponte tem que escutar a interface **vcan0** até a chegada de um quadro, colocar tal quadro dentro de um pacote Ethernet e em seguida enviar o pacote Ethernet para o outro lado da ponte. O Quadro 1 mostra o esqueleto em linguagem de alto nível (C) da primeira parte da abordagem Um-para-um, onde assim que uma mensagem CAN chega ela é enviada dentro de um quadro Ethernet dedicado para o outro lado da ponte.

Quadro 1 – Estratégia Um-para-um - escuta interace CAN e envia para Ethernet

```
1 while(1){
2       /*reading the can frame*/
3       nbytes = read(s_can, &frame, sizeof(struct can_frame));
4
5       /*sending can frame through eth*/
6       if(nbytes > 0){
7       sendto(s_eth, &frame, sizeof(struct can_frame), 0,
8       (struct sockaddr*)&target_host_address, sizeof(struct sockaddr));
9       }
10 }
```

Já na segunda parte da estratégia de encapsulamento Um-para-pm, o lado da ponte que fica responsável por enviar o quadro CAN recebido através da interface Ethernet tem funcionamento oposto, sendo necessário ouvir a interface Ethernet até o recebimento de uma mensagem para então recuperar o quadro CAN através do *payload*. Utilizando a estratégia de Um-para-um, quando uma mensagem Ethernet é recebida são lidos os 16 primeiros bytes referentes ao quadro CAN e uma conversão é feita para a estrutura original do quadro. Esse quadro é por fim escrito no barramento através da interface **vcan1**, completando a comunicação entre as interfaces **vcan0** e **vcan1**. O Quadro 2 mostra como é realizada esta parte do algoritmo.

O algoritmo e o código para a execução em *hardware* físico são exatamente os mesmos, sendo necessário apenas especificar o nome das interfaces físicas no momento de criação dos *sockets* CAN e os endereços IP das interfaces Ethernet. A grande vantagem da utilização das bibliotecas citadas é que o comportamento e configuração das interfaces CAN virtuais são os mesmos de uma interface CAN física. Esta escolha implica na redução do tempo entre a prototipação e a confecção do produto final (visto que economiza-se tempo durante o desenvolvimento) permitindo que sejam realizadas experimentações em ambientes controlados e com custo praticamente nulo de adaptação para o ambiente real.

Quadro 2 – Estratégia Um-para-um - escuta interface ETH e escreve no barramento CAN

```
1 while (1){
2      /*wait for incoming eth message*/
3      length = recvfrom(s_eth, buffer, sizeof(struct can_frame), 0,
4      (struct sockaddr*)&host_address, &hst_addr_size);
5
6        /*cast received bytes to a struct can_frame*/
7      frame = (struct can_frame*)buffer;
8
9        /*write the message to the can if*/
10     if(length > 0){
11      write(s_can, &frame, sizeof(struct can_frame));
12     }
13 }
```

3.2.2 Desenvolvimento

As seguintes abordagens de multiplexação e enfileiramento foram implementadas e avaliadas:

- Um-para-um;
- *Buffer* limitado;
- Contador por prioridade (crédito)
- Gatilho de prioridade;
- **Gatilho por *timeout* do primeiro quadro** (novo)
- **Gatilho estocástico por prioridade** (novo)
- **Gatilho estocástico por ciclo** (novo)

As quatro primeiras estratégias (Um-para-um, buffer limitado (N-1), contador de prioridade e gatilho de prioridade) foram implementadas baseadas em ideias abordadas nos trabalhos relacionados (seção 2.5) estudados para o desenvolvimento deste trabalho. As três últimas estratégias foram inspiradas a partir de diferentes ângulos de análise (como explicado em seção 2.5), discussões e exploração das características que envolvem o barramento CAN e são as principais contribuições deste trabalho em relação à literatura especializada. Esta seção explicará em detalhes como as ideias foram implementadas neste trabalho, tanto as abordagens de outros trabalhos quanto as novas abordagens e o que esperar em relação ao comportamento dessas implementações (disponíveis no GitHub[15]).

[15] https://github.com/ribeirorenan/tcc-multiplexacao-can-ethernet

3.2.2.1 Um-para-um

A abordagem de Um-para-um é o método de enfileiramento mais simples de todos. O algoritmo empacota cada quadro CAN recebido dentro do *payload* de um quadro Ethernet e o envia para a interface Ethernet desejada (método utilizado para exemplificar o funcionamento da comunicação ponto-a-ponto, na subseção 3.2.1.3). Apesar de ser a estratégia mais simples, é o método que consegue enviar as mensagens com o menor atraso possível, pois não é necessário esperar que cheguem outras mensagens, ou realizar operações que considerem a prioridade do quadro CAN antes do envio. A grande desvantagem desta abordagem é o desperdício da largura de banda proporcionada pelo Ethernet, pois o mesmo possui uma capacidade de enviar até 1500 bytes dentro do *payload* sendo que uma estrutura de quadro CAN utiliza apenas 16 bytes.

3.2.2.2 Buffer Limitado (N-1)

A estratégia de *bufferização* dos quadros CAN consiste em receber quadros até uma determinada quantidade, armazenar esses vários quadros dentro de um *buffer* e então enviar esse *buffer* dentro do *payload* Ethernet. A quantidade de quadros a serem alocados dentro de um quadro Ethernet varia de 3 a 93, pois o *payload* tem um valor mínimo de 46 bytes e um valor máximo de 1500 bytes (IEEE 802.3, 2015). Como cada estrutura de quadro CAN ocupa 16 bytes, colocar um número menor do que 3 quadros é um desperdício e o número máximo que é possível alocar em um *payload* Ethernet é de 93 quadros CAN. Esta abordagem não leva em consideração as prioridades dos quadros CAN, apenas os armazena no *buffer* até que seja alcançado a capacidade máxima do *payload*. Sua vantagem é o maior aproveitamento da vazão em potencial quando utilizado o maior número possível de quadros CAN por pacote Ethernet, diminuindo ao máximo o *overhead* de procolo.

3.2.2.3 Contador de Prioridade (Crédito)

A desvantagem do método de *bufferização* é o fato da prioridade das mensagens não ser levada em consideração. Para a implementação da abordagem de contador de prioridade, é utilizada uma variável representado um crédito de tempo para o *buffer* continuar sendo preenchido e que é decrementado de acordo com as prioridades dos quadros CAN que são recebidos mas vão sendo enfileirados. Foram definidos quatro graus de prioridade: baixa, média, alta e muito alta. Quando um quadro de baixa prioridade é recebido, uma unidade de crédito é decrementada no contador; já quando a prioridade é média são decrementadas duas unidades; para alta prioridade o decremento é de três; por fim, para prioridades muito altas são diminuídas quatro unidades do contador. Dessa maneira, se as mensagens que estão sendo encapsuladas têm prioridade alta, o quadro Ethernet será enviado com um número menor de mensagens, com o efeito de que o *buffer* com men-

sagens de alta prioridade vai ocasionar menor atraso de envio do que aquele que possua mensagens com menor prioridade. Ao se utilizar a estratégia do contador de prioridade, a sobrecarga de protocolo pode aumentar mas em contrapartida a ponte começa a ter um comportamento semelhante ao comportamento do barramento CAN.

3.2.2.4 Gatilho por Alta Prioridade

O gatilho por prioridade é a abordagem, dentre aquelas que foram implementadas, que mais se comporta como os mecanismo de priorização do CAN: consiste em fechar o *buffer* de quadros CAN e enviá-lo assim que uma mensagem de alta prioridade é lida no barramento. O gatilho de prioridade poderia até ser implementado em conjunto com as estratégias de *buffer* e de contador de prioridades. A prioridade utilizada para disparar o gatilho de envio pode ser definida de acordo com a semântica das mensagens numa determinada rede CAN, o que possibilita a adaptação do algoritmo para aplicações específicas.

3.2.2.5 Gatilho por *Timeout* do Primeiro Quadro

Esta abordagem, ao contrário do Contador de Prioridade e do Gatilho de Prioridade, busca diminuir o atraso dos quadros CAN levando em consideração os ciclos ao invés da prioridade. A ideia por trás desta implementação é não refazer o trabalho previamente realizado pelo barramento CAN no momento da entrega dos quadros, que irá tratar as prioridades das mensagens recebidas no barramento de destino. Ao invés disso, é definida uma porcentagem máxima de quanto tempo a *primeira mensagem* que foi colocada no *buffer* poderia ser atrasada em relação ao seu ciclo e assim que esse limite é ultrapassado o *buffer* é enviado. Por exemplo, considere que uma mensagem de ID qualquer chega no barramento a cada 50 ms (e é a primeira a ser enfileirada) e, é definido que ela não pode ter um atraso maior do que 50% do seu ciclo. Toda vez que uma nova mensagem for multiplexada no *buffer*, o tempo de espera da primeira mensagem da fila é calculado e é feita a verificação se ela já está enfileirada a mais de 25ms. Quando esse limite é atingido o gatilho é disparado e os quadros CAN enfileirados até aquele momento são todos enviados. A porcentagem máxima de atraso pode ser definida através de uma constante, alterando o comportamento do algoritmo para ser mais ou menos tolerante a atrasos da primeira mensagem no *buffer*.

3.2.2.6 Gatilho Estocástico por Prioridade

Tem como objetivo aplicar o conceito de gatilho de prioridade para o envio de uma mensagem Ethernet sem precisar definir um único limiar fixo para o quadro CAN ser ou não de alta prioridade. Nesta abordagem é sorteado um número uniformemente distribuído entre o ID mínimo e máximo que os quadros CAN podem ter e, se esse número for maior que o ID da mensagem recebida, o *buffer* de quadros CAN é fechado e

a mensagem Ethernet é enviada (quanto menor o ID maior a prioridade, portanto maior será a probabilidade de ativar o gatilho). Durante as simulações percebeu-se que o gatilho para o envio da mensagem Ethernet estava sendo acionado com muita facilidade e para que a estratégia pudesse ter um resultado melhor em relação à vazão, foi feito um ajuste no ID da mensagem recebida, multiplicando-o pela maior quantidade de quadros CAN que cabem em um *payload* Ethernet (93 quadros) dividido por três (a quantidade mínima recomendável de quadros por pacote Ethernet), consistindo em um ajuste encontrado empiricamente mas que proporcionou os melhores resultados durante os testes (ID$*$(93 $\div$3.0)).

3.2.2.7 Gatilho Estocástico por Ciclo

Assim como o gatilho estocástico por prioridade, também utiliza probabilidade para ativar o gatilho de envio da mensagem Ethernet, porém o parâmetro utilizado é o ciclo da mensagem. A ideia de multiplexação por trás do gatilho estocástico por ciclo é a mesma do gatilho por *timeout* do primeiro quadro, não refazer o trabalho já realizado pelo barramento CAN e sim buscar diminuir o atraso das mensagens da melhor maneira possível. Após o cálculo do ciclo de recebimento dos quadros CAN, realizado dinamicamente, quando um novo quadro é compartilhado no barramento CAN e chega até a ponte CAN–ETH, um número uniformemente distribuído entre o menor e o maior ciclo é sorteado e se esse número for maior que o ciclo do novo quadro, o gatilho para o envio da mensagem Ethernet é ativado, dessa maneira quanto menor o ciclo maior a probabilidade de ativar o gatilho. Para que as mensagens CAN não fossem enviadas precocemente com poucos quadros foi necessário realizar um ajuste empírico, multiplicando o ciclo por valores que variaram de 0.04 a 1.5 vezes a quantidade máxima de quadros CAN por mensagem Ethernet (93 quadros). Nesta abordagem foi difícil escolher um valor fixo para o ajuste, por causa das diferentes características que eram obtidas durante os testes. O melhor valor de ajuste encontrado foi de 1 vezes a quantidade máxima de quadros CAN por mensagem Ethernet.

3.2.3 Experimentação

A seguir é descrito como foi conduzida a experimentação em bancada para a realização dos protótipos e prova de conceito.

3.2.3.1 Arduino

Comunicação CAN

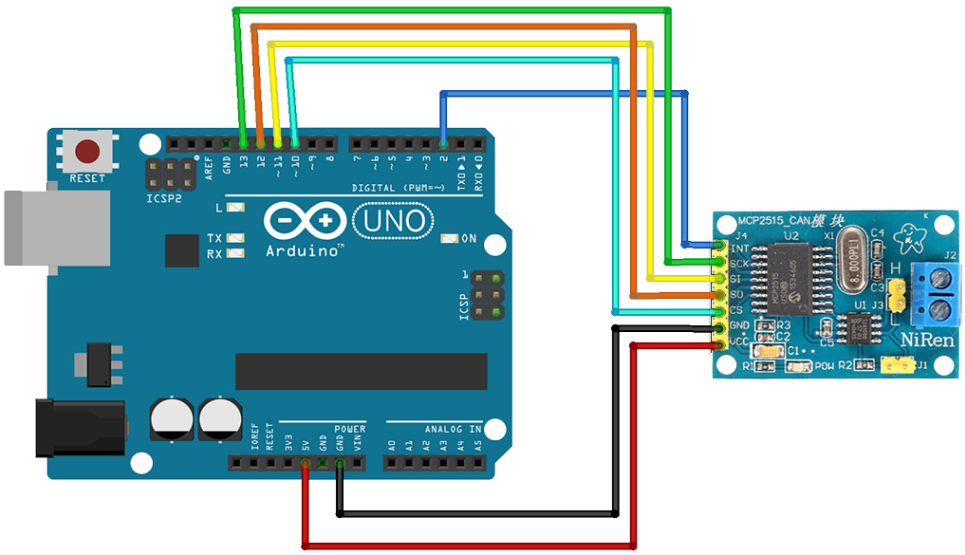
A primeira etapa neste ponto do desenvolvimento do projeto foi o estudo de bibliotecas do Arduino para a utilização dos transceptores CAN MCP2515 (Figura 12). Para

realizar a ligação do módulo CAN ao Arduino a montagem foi feita da seguinte maneira:

- Pinos VCC (5 V) e GND do módulo CAN nas respectivas entradas do Arduino;
- *Slave* (SS) no pino de entrada 10;
- *Master Out Slave In* (SI) e *Master In Slave Out* (SO) nas entradas 11 e 12 respectivamente;
- *Serial Clock* (SCK) no pino de entrada 13;
- E por fim o pino INT do módulo ao pino de entrada 2 do Arduino.

Foram exploradas diversas bibliotecas, das quais escolhemos a `MCP_CAN_lib`[16], que possibilitou a comunicação entre dois nós CAN (cada um utilizando um Arduino como microcontrolador) com praticidade. Os exemplos mais simples dessa biblioteca incluíam um exemplo de *Send*, que envia um quadro CAN através do barramento, um exemplo de *Receive* que escuta o barramento e recebe um quadro CAN e também um exemplo de *Bridge* que possibilita a comunicação entre dois barramentos distintos. Esses exemplos foram testados com os Arduinos Mega e Pro Mini em conjunto com os transceptores MCP2515, finalizando a realização da primeira etapa de comunicação básica em um barramento CAN utilizando Arduino.

Figura 12 – Ligação entre Arduino e Módulo CAN



Fonte: (CORE, 2017)

[16] GitHub: https://github.com/coryjfowler/MCP_CAN_lib)

Comunicação Ethernet

Após a etapa de comunicação entre transceptores CAN ter sido bem sucedida, foi dado início nos estudos sobre as bibliotecas para comunicação Ethernet-para-Ethernet utilizando microcontroladores Arduino. Como os Arduinos utilizados não possuem nativamente interface Ethernet, utilizamos *shields* Wiznet 5100 e módulos encj28[17] como interface Ethernet para nossos microcontroladores. O shield utilizado funcionou perfeitamente, possibilitando a troca de mensagens utilizando a interface Ethernet em um computador, porém com o módulo encj28 disponível, não foi possível realizar a transmissão de pacotes Ethernet junto ao arduino e em decorrência da dificuldade encontrada por não identificarmos a fonte do problema (defeito no módulo ou incompatibilidade com o arduino). Foram comprados novos módulos Ethernet e esse estudo de Ethernet-para-Ethernet foi iniciado com o Raspberry Pi.

3.2.3.2 Raspberry Pi

Comunicação CAN

O Raspberry possui nativamente pinos de entrada e saída de propósito geral (GPIO). Através destes pinos é possível conectar os transceptores CAN MCP2515 e realizar transmissão de dados utilizando interface de comunicação SPI. Efetuando o interfaceamento correto entre os pinos de entrada e saída dos Raspberry Pi e a interface SPI do transceptor CAN MCP2515 (e carregando os módulos referentes ao CAN para dentro do kernel Linux), foi possível realizar a comunicação entre o Raspberry e os transceptores. A Figura 13 mostra um Raspberry Pi conectado a um módulo CAN através da interface SPI.

Na Figura 13 o módulo CAN está sendo alimentado pelo pino de 3,3 V do Raspberry Pi e apesar do MCP2515 funcionar com a alimentação de 3,3 V, o transceptor CAN necessita de 5 V para que a comunicação entre o módulo e o barramento CAN funcione corretamente. No início do desenvolvimento da implementação do protótipo não foi possível realizar a comunicação entre o Raspberry Pi e o barramento CAN de maneira satisfatória em razão dos pinos de entrada e saída do Raspberry Pi suportarem no máximo 3,3 V e, se o módulo CAN fosse alimentado com a tensão necessária para seu funcionamento correto (5 V), os Rasperrys Pi poderiam ser danificados. Para contornar este problema foram adquiridos conversores de nível lógico (Figura 11), que têm o papel de fazer a conversão de tensão necessária entre os pinos de entrada e saída do Raspberry e os pinos SPI do módulo CAN. Entretanto quando tivemos acesso aos conversores e realizamos o interfaceamento da maneira desejada entre o Raspberry Pi e o módulo CAN, já não havia tempo hábil para a montagem do protótipo final que é apresentado na **??**. Por

[17] http://www.microchip.com/wwwproducts/en/en022889

Figura 13 – Raspberry Pi + Módulo CAN MCP2515

Fonte: Autor

este motivo os testes foram todos feitos utilizando os ambientes de simulação descritos no decorrer do trabalho.

Comunicação Ethernet

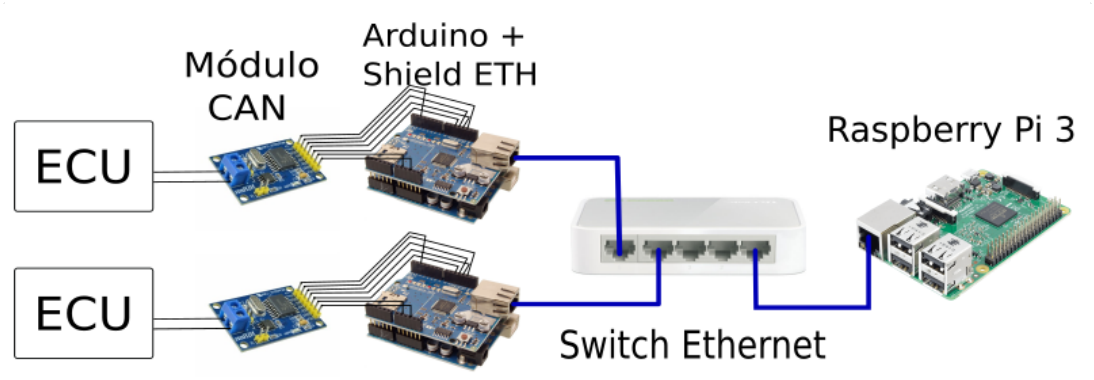
Como o Raspberry Pi é um computador com arquitetura ARM, é possível utilizar sistemas operacionais que sejam compatíveis com o Raspberry Pi (e sua arquitetura). Dentre os vários sistemas operacionais disponíveis foi utilizado o Raspbian[18], que é uma distribuição Linux baseada no Debian. Em decorrência da facilidade por existir um sistema operacional para a interação com o usuário, não foi necessária a busca de nenhuma biblioteca externa às bibliotecas nativas do Raspbian (para realizar a comunicação entre interfaces Ethernet), que dispunha de todas as ferramentas de *software* utilizadas. Além da comunicação entre Raspberrys Pi, também foram realizadas comunicação entre Raspberry e notebooks através da interface Ethernet nativa dos mesmos (ambos os dispositivos utilizando alguma distribuição Linux). As ferramentas utilizadas para testar as conexões Ethernet – Ethernet neste primeiro momento foram o `ifconfig` na configuração das interfaces e o `ping` para assegurar o conectividade de ida e volta das mensagens, ambas tipicamente predisponibilizadas no Sistema Operacional.

[18] https://www.raspbian.org/

Ponte CAN - Ethernet - CAN

Uma das maneiras de realizar uma ponte CAN – ETH – CAN é utilizar um *switch* Ethernet para interconectar as ECUs (ou o *hardware* responsável por gerar mensagens CAN). A Figura 14 mostra um exemplo desse tipo de ponte interconectando duas ECUs. A comunicação funciona da seguinte maneira: Um Arduino recebe mensagens de um módulo CAN (conectado a uma ECU, barramento ou até mesmo gerador de pacotes CAN) e empacota esse quadro dentro de uma mensagem Ethernet que é enviada através do *switch* para o Raspberry. O Raspberry por sua vez recebe os quadros CAN através da própria interface Ethernet e, de acordo com o acúmulo de quadros CAN, aplica alguma estratégia de multiplexação para entregar as mensagens para cada um dos módulos CAN. Como o CAN é um barramento, quando o Raspberry enviar de volta os quadros CAN multiplexados em uma mensagem Ethernet, é necessário endereçar a mensagem Ethernet para todas as interfaces CAN, bastando enviar no modo *broadcast* para que o próprio *switch* Ethernet entregue os quadros multiplexados dentro do *payload* da mensagem Ethernet para todos os Arduinos que irão escrever os quadros CAN de volta na interface CAN e, consequentemente, no barramento.

Figura 14 – Ponte CAN - ETH - CAN Utilizando Switch



Fonte: Autor

Vale observar que nos protótipos de Ethernet Automotiva já existentes são utilizados um par de fios trançados (IEEE 802.3bw, 2015), enquanto no protótipo descrito foi utilizado um enlace Fast Ethernet (IEEE 802.3u, 1995), com cabo UTP de quatro pares de fios e conectores RJ-5 [19].

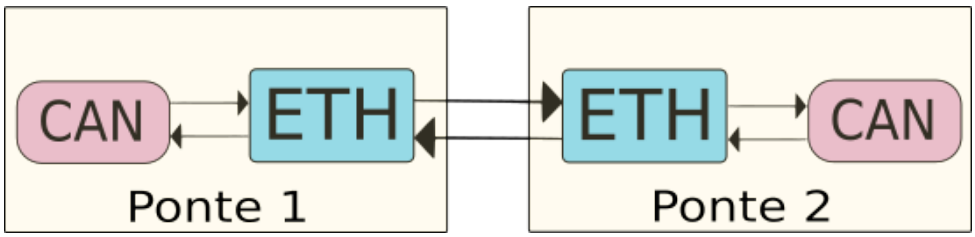
[19] https://en.wikipedia.org/wiki/RJ45

3.2.4 Validação

3.2.4.1 Estratégias de multiplexação

Todas as estratégias de multiplexação foram testadas e validadas através de simulação seguindo o esquema mostrado na Figura 15.

Figura 15 – Esquema de simulação de estratégias



Fonte: Autor

Foram iniciadas duas instâncias de "pontes". Ambas as instâncias possuem um *socket* que realiza comunicação com uma interface CAN e outro *socket* que realiza comunicação com uma interface Ethernet. Foram gerados diversos tráfegos CAN utilizando o `cangen` (ferramenta fornecida pelo SocketCAN e `can-utils`) e à medida que uma das pontes recebia pacotes CAN, a estratégia que estava sendo validada ia realizando a multiplexação, de maneira que o pacote Ethernet com os quadros CAN em seu *payload* era enviado pelo *socket* Ethernet para o outro lado da ponte. Quando a interface Ethernet da segunda ponte lia no *socket* o pacote Ethernet, os quadros CAN eram recuperados e então escritos individualmente no barramento CAN pelo *socket* da interface CAN desta ponte.

Para validar a comunicação, foram utilizadas as ferramentas `candump`, `cansniffer` e `canbusload` (fornecidas pelo SocketCAN e `can-utils`). Através do `candump` e do `cansniffer` foi verificado que todas as mensagens que eram enviadas em uma interface CAN sem perdas (de um lado da ponte) eram recebidas também na outra interface CAN (do outro lado da ponte). Com a utilização do `canbusload` foram verificadas as cargas de utilização dos barramentos que foram interligados pela ponte CAN–ETH–ETH–CAN.

3.2.4.2 Simulação de Tráfego CAN

Como foi dito na subseção 3.2.4.1 foi utilizado o `cangen` para gerar simulações de tráfegos CAN, porém essa ferramenta possui a limitação de somente gerar tráfego de mensagens com IDs e ciclos aleatórios ou mensagens com IDs e/ou ciclos fixos, não disponibilizando opção alguma de gerar um tráfego com mais de um ID e ciclo pré-definidos. Para contornar esse problema foi criada uma tabela de IDs e ciclos manualmente e para cada ID e Ciclo foi disparado um `cangen`, de maneira concorrente (em paralelo). Assim foi

possível criar simulações de tráfegos com IDs e ciclos predeterminados, que proporcionaram A repetibilidade dos testes em um ambiente controlado. Juntamente com o `cangen`, também foi utilizado o `canbusload` para verificar a porcentagem de utilização do barramento. Através destas configurações para o cangen foram gerados tráfegos de 10%, 50%, 75% e 100% de utilização de um barramento CAN.

4 Resultados e Análise

4.1 Comportamento das Estratégias em Relação Ao Atraso

Nesta seção serão apresentados os gráficos obtidos através dos testes das estratégias de multiplexação com aproximadamente 100% de utilização do barramento (saturado através de *logs* criados empiricamente) e como cada estratégia se comportou em relação aos atrasos de comunicação gerados pela intermediação dos algoritmos.

Conforme explicado na metodologia, os algoritmos foram testados por meio de simulação de um barramento CAN, cujo *log* continha o envio de dezenas de milhares de mensagens CAN, referentes a centenas de CAN IDs diferentes (escolhidos de maneira aleatória, porém fixos no log para que se pudesse repetir os testes). Portanto, para apresentar os resultados de maneira tangível, foram calculadas estatísticas descritivas dos resultados obtidos por cada estratégia, apresentados nas seções abaixo sob forma de gráficos, tabelas e sua análise.

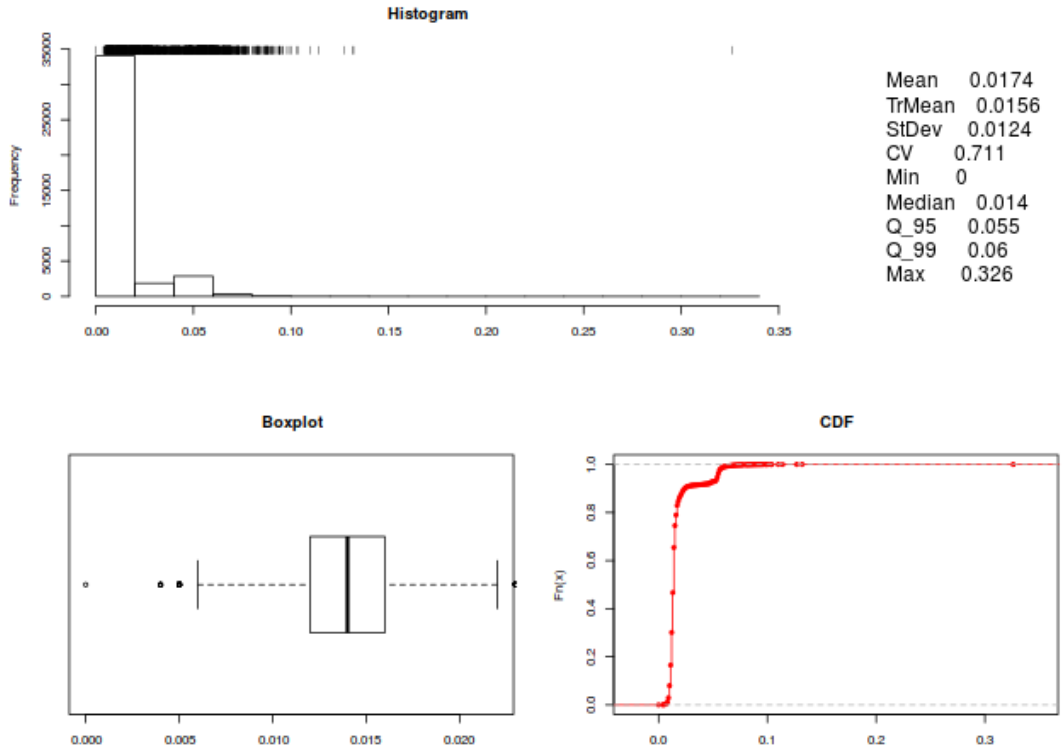
Nos resultados abaixo, para cada estratégia implementada e simulada são apresentadas as seguintes visões dos atrasos gerados pelo enfileiramento e multiplexação. A figura apresentando as **estatísticas descritivas** (histograma, *boxplot*, função de probabilidade acumulada CDF) foi gerada a partir de todos os valores absolutos dos atrasos visando caracterizar a distribuição dos atrasos, sua variabilidade e probabilidades. Já as figuras com **gráfico de barras** (e linha de regressão linear), foram geradas a partir do valor médio dos atrasos para cada ID CAN e seu respectivo ciclo de envio de mensagem. Portanto, não deve-se buscar relação direta de valores entre figuras desses dois tipos diferentes, visto que na primeira há atrasos absolutos e na segunda, atrasos médios.

4.1.1 Um-para-um

Sobre os valores absolutos de todos os atrasos registrados durante a simulação, conforme estatísticas apresentadas na Figura 16, a média dos atrasos absolutos para esta estratégia foi de apenas 0,0174 ms (17,4 μs), com um desvio padrão de 0,0124 (12,4 μs) e com 95% das mensagens experimentando atrasos abaixo de 0,06 ms (60 μs). Tais atrasos podem ser considerados como baixos (e provavelmente aceitáveis) para boa parte das aplicações que fazem uso do barramento CAN. Observe que o atraso gerado por esta estratégia é muito baixo pois consiste apenas no tempo gasto para o encapsulamento de uma única mensagem.

Entretanto, o custo de enviar uma mensagem CAN sozinha em um pacote Ethernet é a baixa eficiência no uso do protocolo subjacente, de maneira que apenas o cabeçalho Ethernet já é maior que uma única mensagem CAN pelo qual é responsável. O próprio *payload* Ethernet tem um tamanho mínimo obrigatório de três vezes o tamanho de uma mensagem CAN típica. Esta discussão da eficiência no uso do protocolo Ethernet será apresentada na próxima seção, após terem sido apresentados os resultados dos atrasos causados por cada estratégia de enfileiramento e multiplexação.

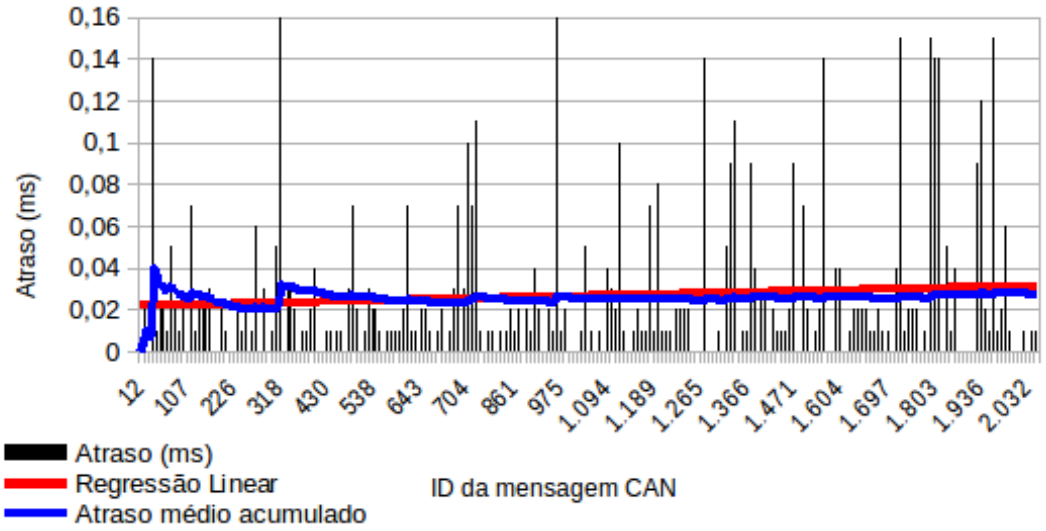
Figura 16 – Estatísticas dos atrasos (ms) - Um-para-um



Fonte: Autor

Do ponto de vista da média dos atrasos para cada CAN ID, como pode ser observado na Figura 17, os atrasos por CAN ID ficaram em média abaixo de 0,03 ms, com 95% dos CAN IDs experimentando atrasos abaixo de 0,13 ms, em média. Vale ressaltar que o ID que experimentou um atraso médio maior que os demais IDs foi de 0,40 ms.

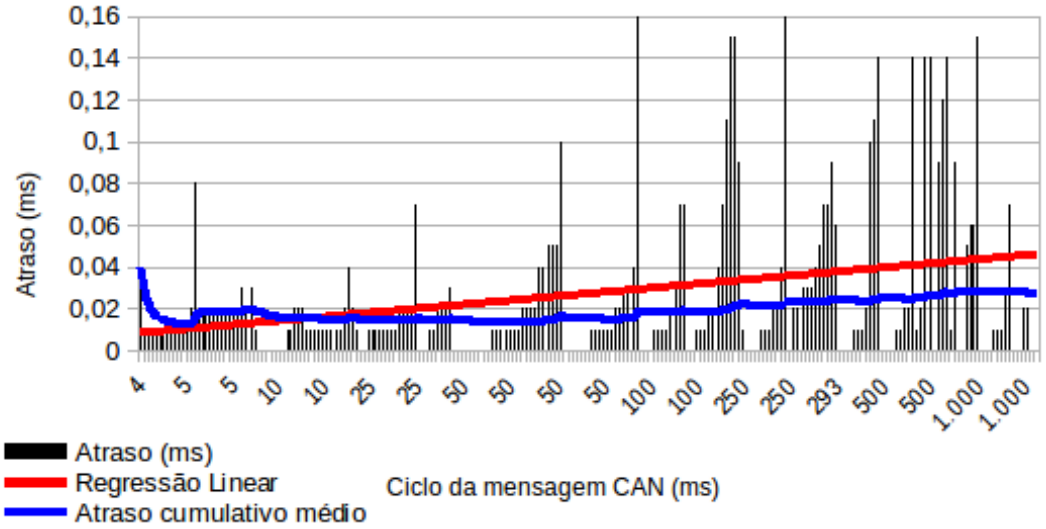
Figura 17 – Um-para-um (ID x Atraso)



Fonte: Autor

Para muitas das aplicações de CANbus, tais atrasos causados pelo enfileiramento, multiplexação e ponte CAN-Ethernet podem ser considerados viáveis. Observando a Figura 18, as mensagens com ciclo mais longo apresentaram um ligeiro acréscimo no atraso observado, o que pode ser explicado pelo próprio processo de arbitragem para acesso ao barramento CAN.

Figura 18 – Um-para-um (Ciclo x Atraso)

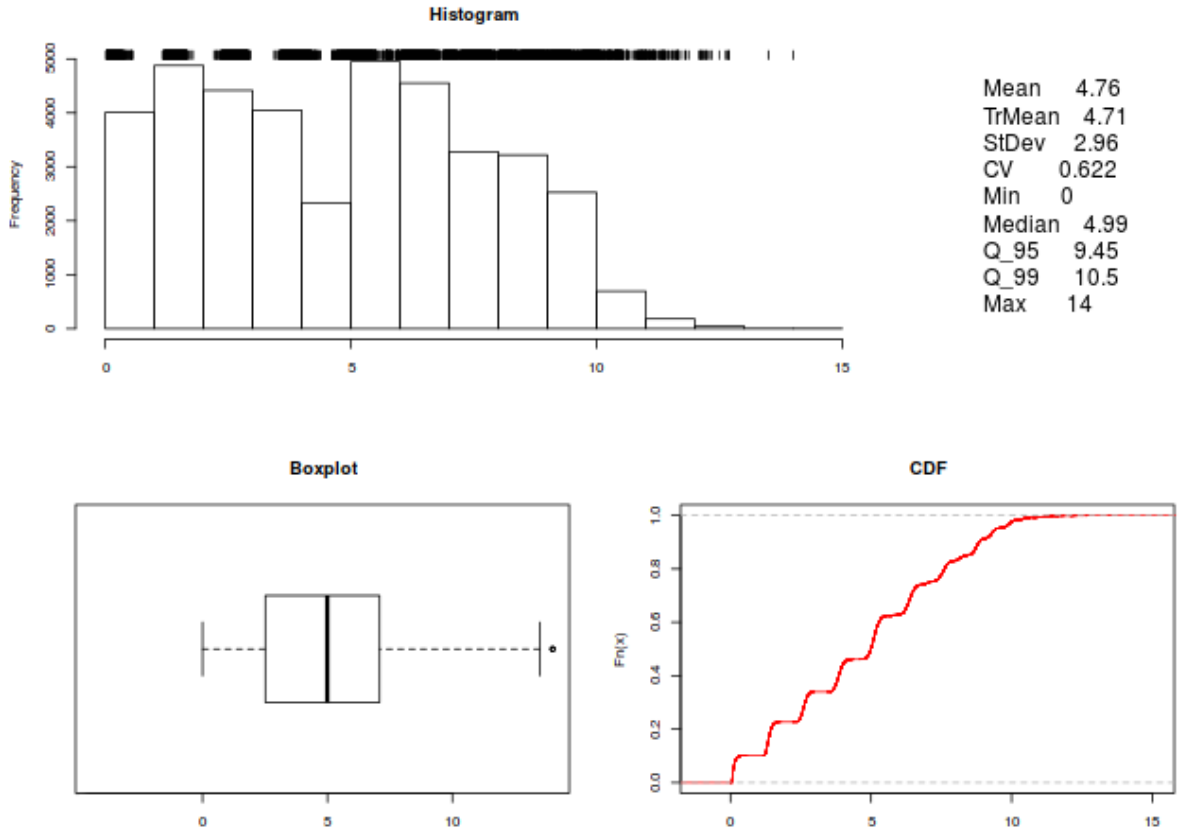


Fonte: Autor

4.1.2 Buffer Limitado (N-1)

Ao observar a Figura 19, percebe-se que o atraso gerado pela estratégia de Buffer Limitado (N-1) (KERN et al., 2011) causa em média 4,76 ms, chegando a até 14 ms na simulação. Tal nível de atraso por enfileiramento é o maior observado durante as simulações realizadas neste trabalho, para o cenário de 100% de carga do barramento CAN. Isso se dá pelo fato das mensagens CAN serem enfileiradas até que contabilizem 93, ou seja, a quantidade máxima de mensagens que caberia em um quadro Ethernet (utilizando no máximo 1488 bytes do **payload** Ethernet, pois cada quadro CAN têm fixo 16 bytes na implementação deste trabalho). Vale notar que o atraso causado por essa técnica de enfileiramento foi estatisticamente o pior dentre as estratégias avaliadas.

Figura 19 – Estatísticas dos atrasos (ms) - Buffer Limitado (N-1)

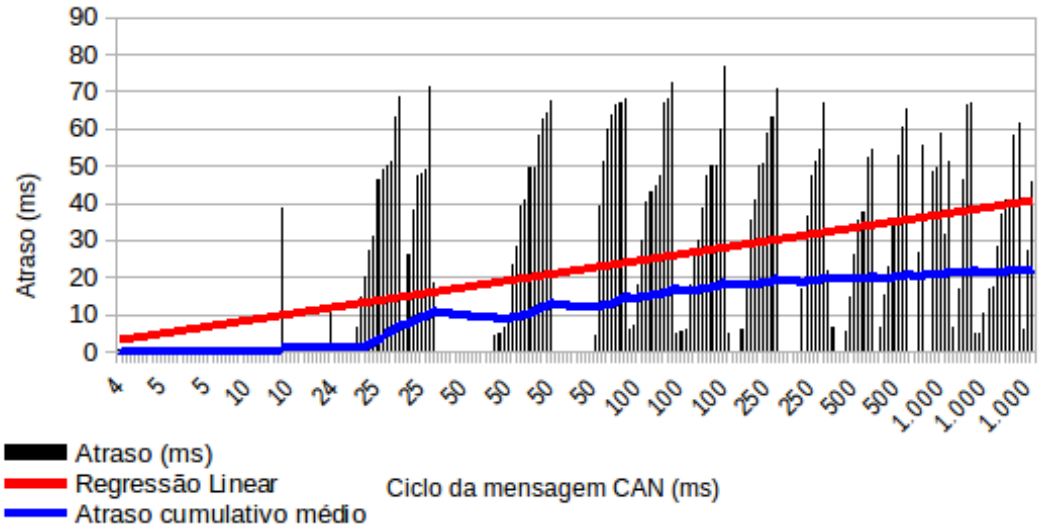


Fonte: Autor

Comparando a Figura 20 com a Figura 21 é importante ressaltar que, apesar de não ter sido observado padrão de comportamento do atraso por ID, ao analisar o gráfico de **Atraso x Ciclo** percebe-se que as mensagens com maior ciclo sofreram o menor atraso (vide linha de regressão). Isso ocorre pelo fato de os quadros com maior ciclo terem menor incidência no barramento (2 vezes por segundo no caso de ciclos de 500 ms), enquanto

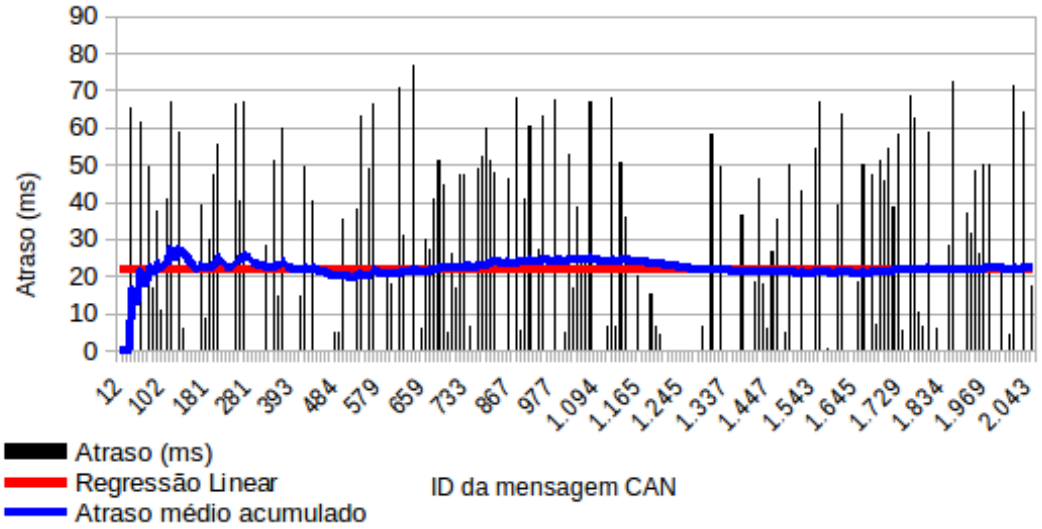
os quadros de menor ciclo são escritos no barramento com maior frequência (200 vezes por segundo no caso de ciclos de 5 ms). Assim, dependendo de quão baixo for o ciclo do quadro CAN, nesta estratégia pode ser que mais de um quadro com o mesmo ID seja enfileirado no *buffer*, antes de serem ambos despachados em um único pacote Ethernet.

Figura 20 – Buffer Limitado (N-1) (Ciclo x Atraso)



Fonte: Autor

Figura 21 – Buffer Limitado (N-1) (ID x Atraso)

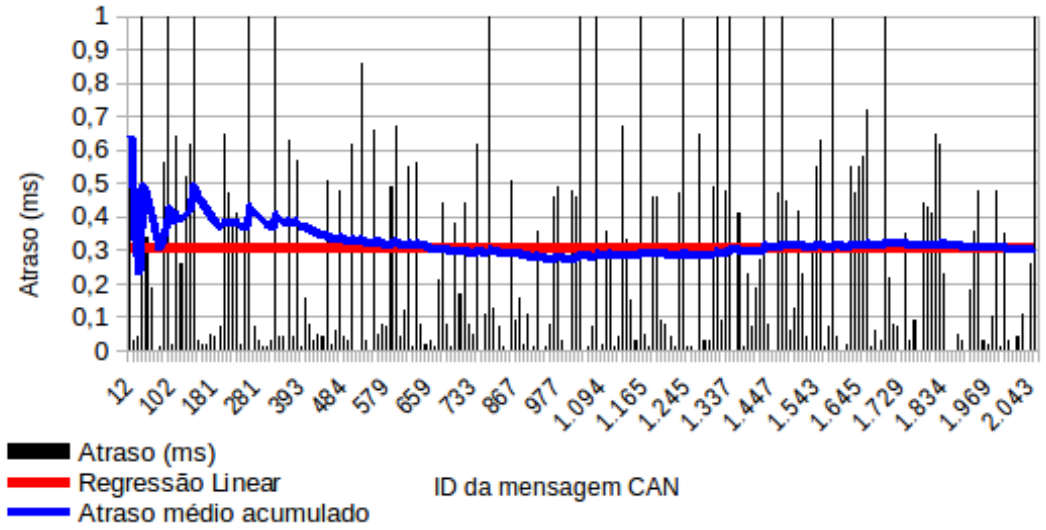


Fonte: Autor

4.1.3 Contador de Prioridade (Crédito)

Apesar desta estratégia ser inspirada no comportamento de arbitragem do barramento CAN e dar prioridade para as mensagens com menor ID, percebemos através da Figura 22 que a utilização de tal estratégia, nesta simulação, não causou grande impacto na redução do atraso em relação à prioridade dos quadros (vide linha de regressão).

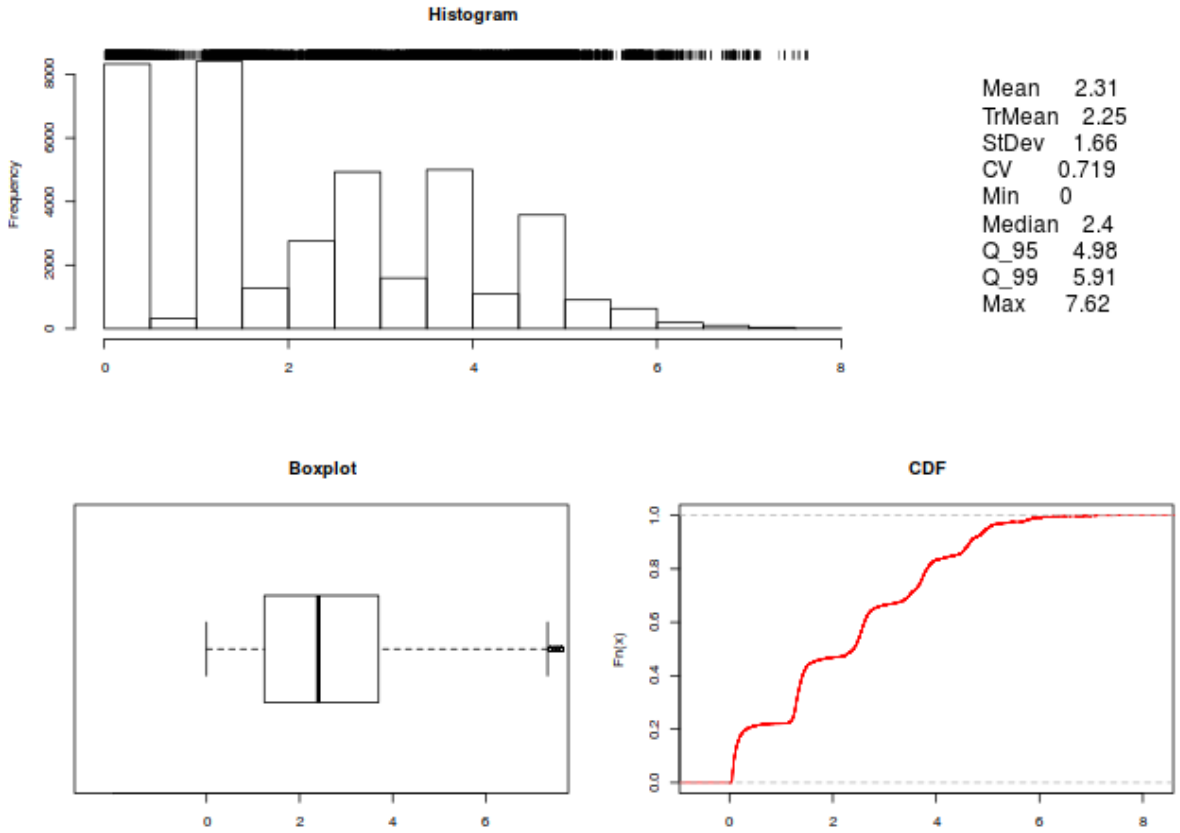
Figura 22 – Contador de Prioridade (Crédito) (ID x Atraso)



Fonte: Autor

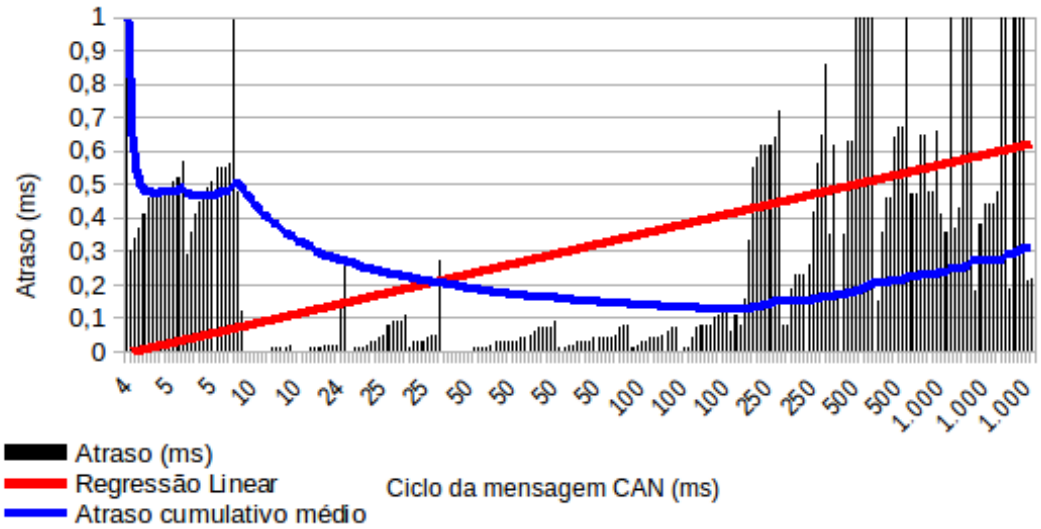
Porém, percebemos pela Figura 24 e Figura 23 que os atrasos (absoluto e médio, respectivamente) foram menores do que aqueles causados pela estratégia de utilização de *Buffer* Limitado (KERN et al., 2011), obtendo uma média de 2,31 ms. Ou seja, apesar de (aparentemente) não mimetizar perfeitamente o comportamento de arbitragem do CAN, a estratégia de créditos por prioridade conseguiu reduzir pela metade o atraso médio e máximo observados.

Figura 23 – Estatísticas dos atrasos (ms) - Contador de Prioridade (Crédito)



Fonte: Autor

Figura 24 – Contador de Prioridade (Crédito) (Ciclo x Atraso)

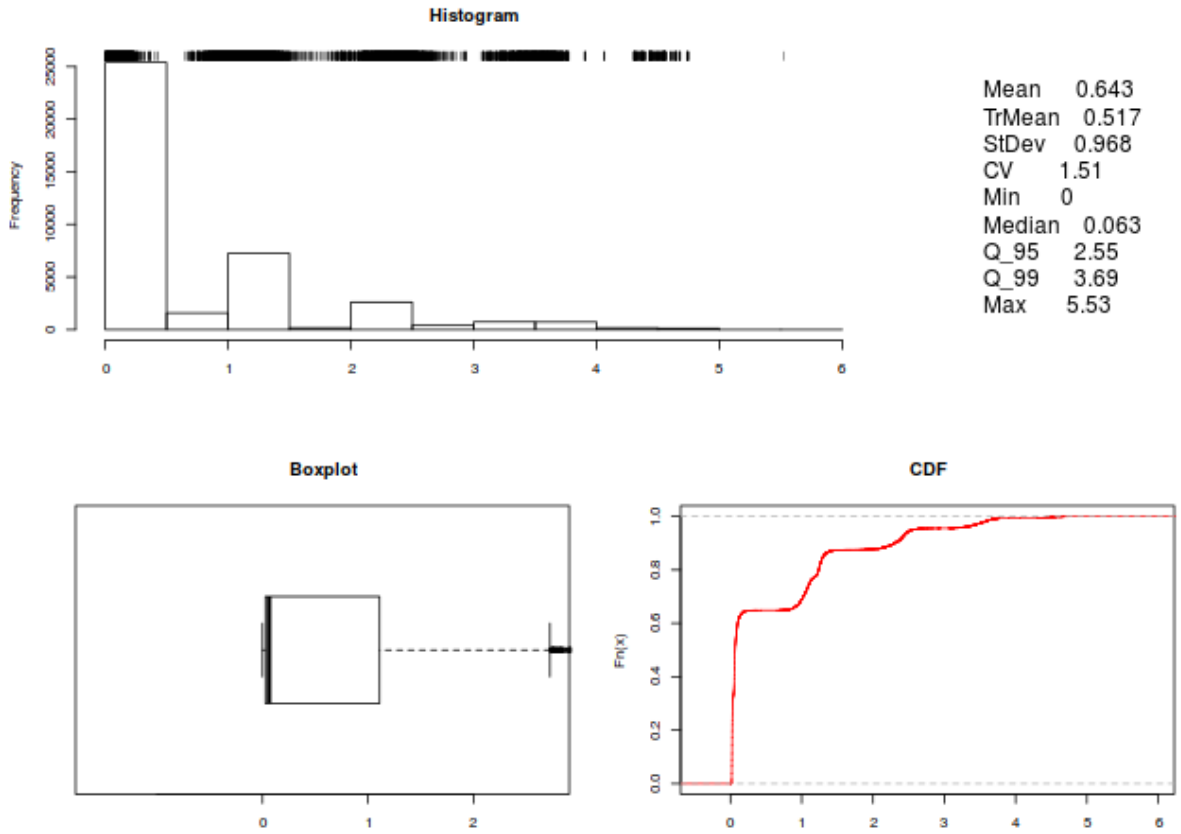


Fonte: Autor

4.1.4 Gatilho por Alta Prioridade

Analisando a Figura 25, percebemos que o atraso do gatilho de prioridade (KERN et al., 2011) obteve resultados ainda melhores que a estratégia de contador de prioridades, com atraso médio de 0,643 ms e mediano de 0,063 ms. Por conta da dicotomia na segregação das mensagens, o coeficiente de variação (1,51) registrado foi o dobro das anteriores, mostrando uma clara heterogeneidade nos atrasos causados pela estratégia de enfileiramento e multiplexação. Faz-se necessário ressaltar que, para utilizar esta estratégia de enfileiramento deve-se definir exatamente qual é o limiar de decisão para segregar quadros com alta e baixa prioridade no enfileiramento, decisão importante para garantir o desempenho desejado pela aplicação. Para fins didáticos, na simulação deste trabalho foi configurado como 256 o ID a partir do qual as mensagens CAN são tratadas pela ponte como sendo de baixa prioridade, ou seja, 1/8 do espaço de endereçamento dos 2048 possíveis IDs.

Figura 25 – Estatísticas dos atrasos (ms) - Gatilho por Alta Prioridade

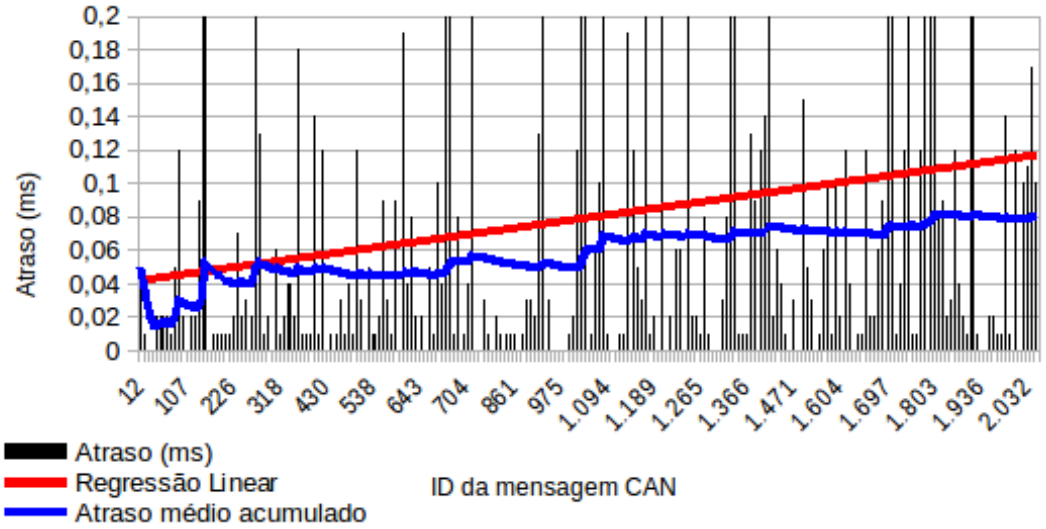


Fonte: Autor

É necessário ressaltar que a Figura 26 é apresentada com o eixo vertical limitado em até 0,2 ms de atraso, em contraste com a Figura 27. O objetivo foi destacar o fato de

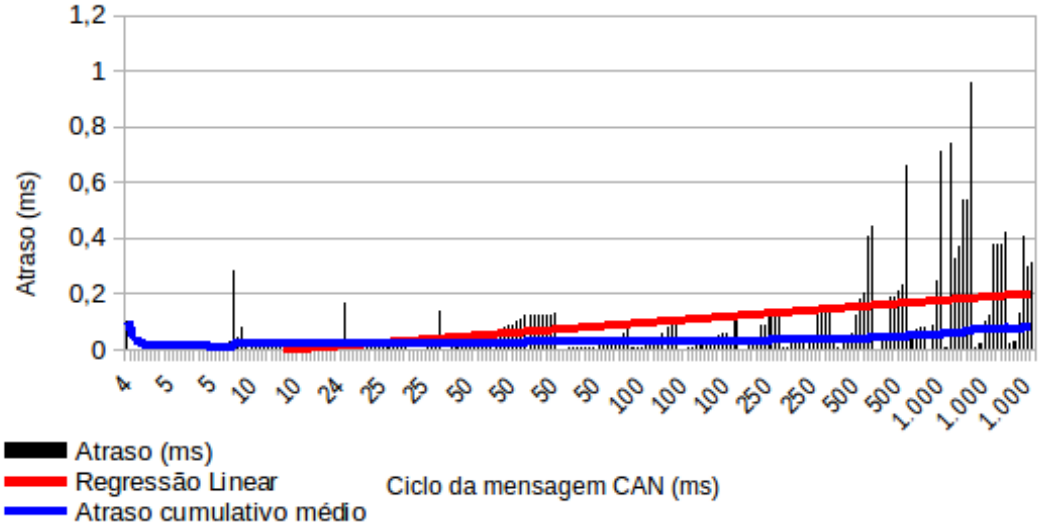
que todos os quadros que possuem ID de alta prioridade (ID < 256) tiveram atrasos bem menores (abaixo de 0,03 ms), ao passo que o restante das mensagens (ID > 256) sofreram com atrasos até 30 vezes maiores.

Figura 26 – Gatilho por Prioridade (ID x Atraso)



Fonte: Autor

Figura 27 – Gatilho por Prioridade (Ciclo x Atraso)

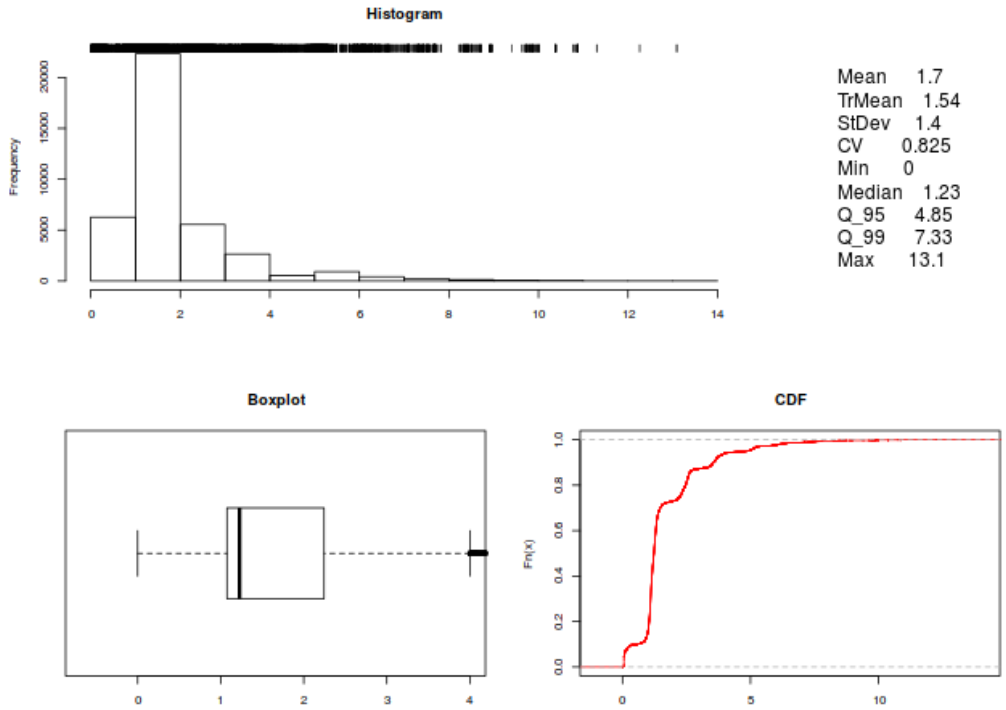


Fonte: Autor

4.1.5 Gatilho por *Timeout* do Primeiro Quadro

O objetivo desta abordagem é diminuir o atraso das mensagens CAN baseando-se em seus ciclos. O atraso médio observado na utilização desta estratégia foi de 1,7 ms (Figura 28), bem abaixo do observado na estratégia de Contador de Prioridade, seja para o atraso médio, atraso mediano ou atraso máximo.

Figura 28 – Estatísticas dos atrasos (ms) - Gatilho por *Timeout* do Primeiro Quadro

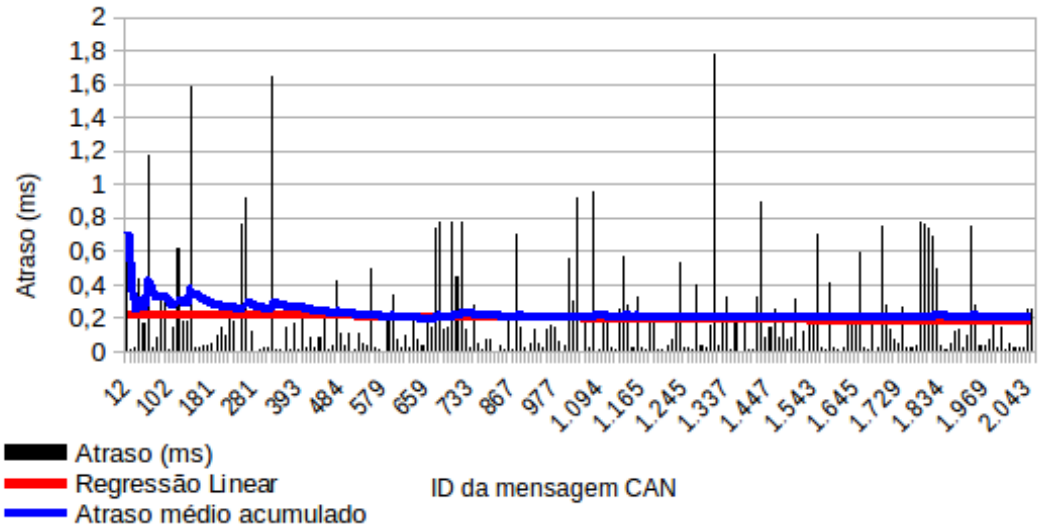


Fonte: Autor

Apesar de não ter sido a estratégia com os menores atrasos, analisando a Figura 29, pode-se observar que a maioria das mensagens CAN sofreu atrasos menores que 1,23 ms. Isso consiste em um aumento relativo de 25% no atraso para uma mensagem com ciclo de apenas 5 ms.

Pela Figura 30, percebemos que os atrasos foram distribuídos de acordo com o ciclo dos quadros CAN, o que era esperado mas consiste em uma característica muito interessante, pelo fato que se segue. Quadros com ciclos maiores observaram atrasos absolutos maiores, mas sofrem relativamente menor penalidade do que quadros com ciclos menores. Por exemplo, 5 ms de atraso em um quadro que tem 5 ms de ciclo é inaceitável (100% atrasado) pois invadiria o próximo envio da mensagem. Porém, em uma mensagem que possui 1000 ms de ciclo os mesmos 5 ms de atraso seriam provavelmente toleráveis (0,5% atrasado).

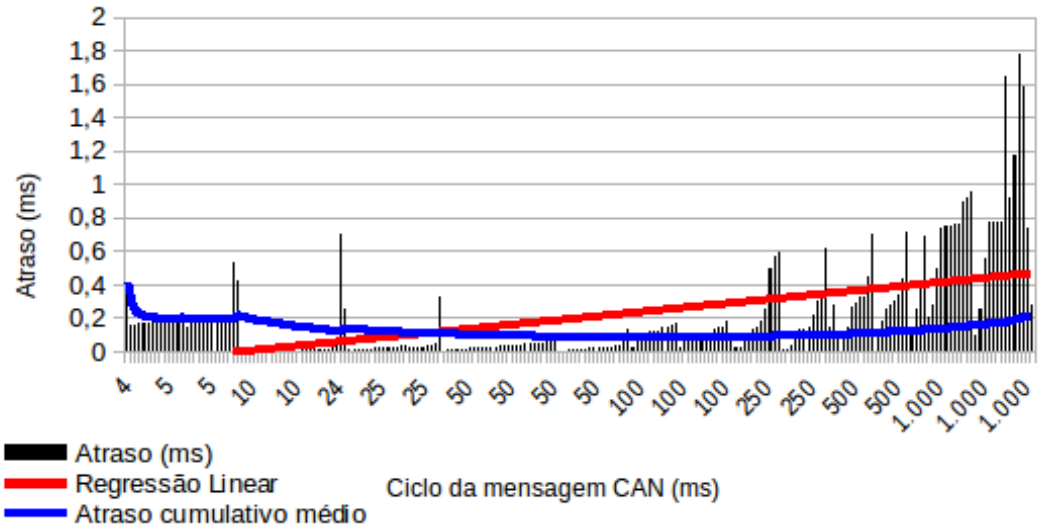
Figura 29 – Gatilho por *Timeout* do Primeiro Quadro (ID x Atraso)



Fonte: Autor

Observe na Figura 30 que houve um impacto maior dos atrasos de enfileiramento para as mensagens com ciclos de envio abaixo de 10 ms. Considere que, caso a primeira mensagem a ser enfileirada tenha um ciclo diferente das demais, tais mensagens podem ser prejudicadas (ou beneficiadas) em função da ordem de magnitude do *timeout* da primeira mensagem da fila, uma vez que o despacho do pacote Ethernet conterá todas as mensagens enfileiradas até o estouro desse *timeout*.

Figura 30 – Gatilho por *Timeout* do Primeiro Quadro (Ciclo x Atraso)

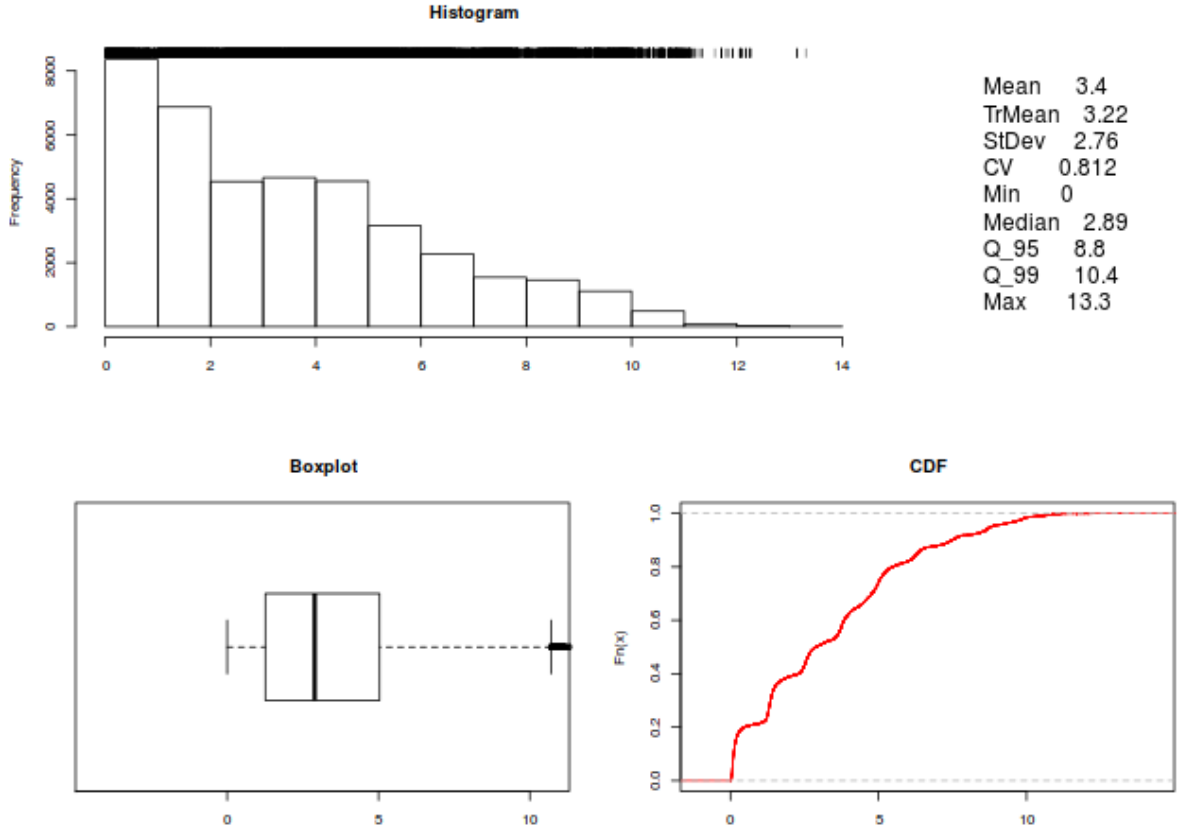


Fonte: Autor

4.1.6 Gatilho Estocástico por Prioridade

Também inspirado na priorização por ID presente na rede CAN, esta estratégia tem o objetivo de utilizar um sorteio a cada nova chegada de mensagem CAN à fila, com maior probabilidade de mensagens de alta prioridade dispararem o gatilho e despacharem toda a fila no pacote Ethernet. Esta estratégia obteve um atraso médio de 3,4 ms porém com mediana de 2,89 ms (vide Figura 31).

Figura 31 – Estatísticas dos atrasos (ms) - Gatilho Estocástico por Prioridade

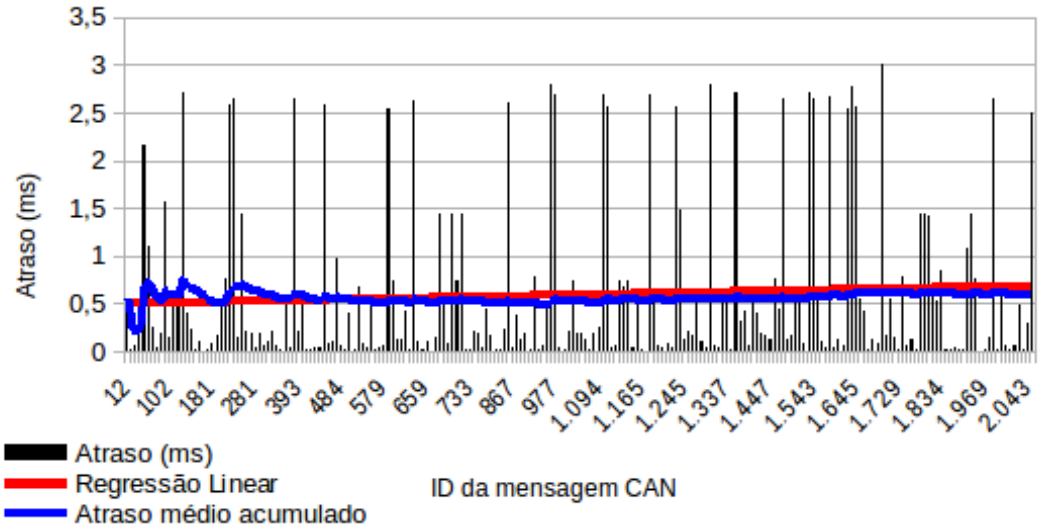


Fonte: Autor

Observando o gráfico de IDxAtraso (Figura 32), não percebe-se tão explicitamente um comportamento orientado ao ID como foi visto nos resultados da estratégia de gatilho de prioridade (Figura 26). Entretanto, é explicitado na Figura 33 a consequência de tal mecanismo para as mensagens CAN de ID com baixa prioridade mas com maior frequência na tentativa de envio (ciclo menor que 10 ms).

A partir dos resultados apresentados na Figura 33, tal penalização de mensagens com ciclo curto mas ID de menor prioridade deve ser mitigada caso deseje-se que a ponte CAN-Ethernet atenda aos requisitos da aplicação quanto ao atraso de enfileiramento,

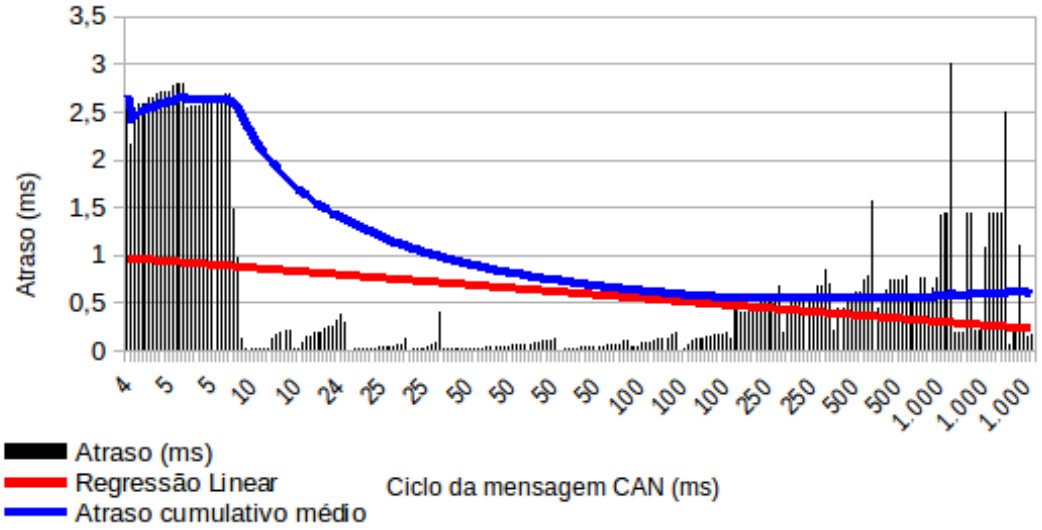
Figura 32 – Gatilho Estocástico por Prioridade (ID x Atraso)



Fonte: Autor

variação deste atraso e taxa de perda, questões importantes para aplicações multimídia que utilizem o barramento CAN para trafegar dados entre ECUs.

Figura 33 – Gatilho Estocástico por Prioridade (Ciclo x Atraso)



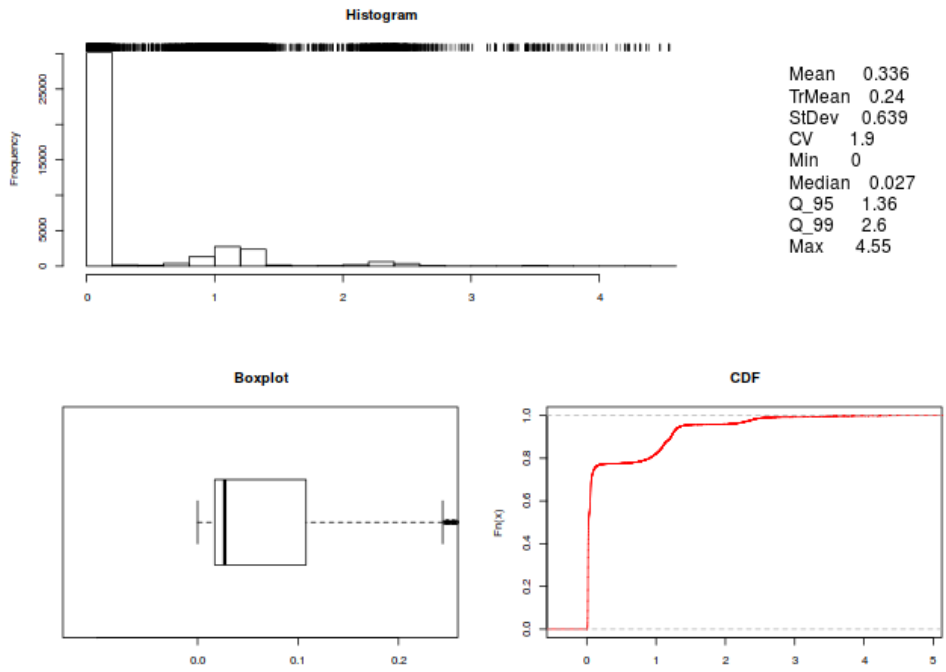
Fonte: Autor

4.1.7 Gatilho Estocástico por Ciclo

Considerando os pontos fortes e fraquezas das estratégias acima analisadas, uma proposta promissora é a que também utiliza sorteio, mas com probabilidade maior de disparo de gatilho para mensagens que possuem ciclo menor. Tal abordagem de enfileiramento e multiplexação é proposta neste trabalho e busca distribuir os atrasos de acordo com os ciclos dos quadros CAN, reduzindo os atrasos para as mensagens de menores ciclos por serem elas as que mais seriam afetadas pela magnitude dos atrasos.

Ao observar na Figura 34 o atraso de enfileiramento causado por esta nova estratégia, percebemos uma média global de apenas 0,336 ms e 99% das mensagens CAN sendo enfileiradas e despachadas com um atraso menor que 2,6 ms. Estatisticamente, esta estratégia de enfileiramento obteve os menores atrasos em comparação com as outras implementações, excetuando-se a estratégia Um-para-um.

Figura 34 – Estatísticas dos atrasos (ms) - Gatilho Estocástico por Ciclo

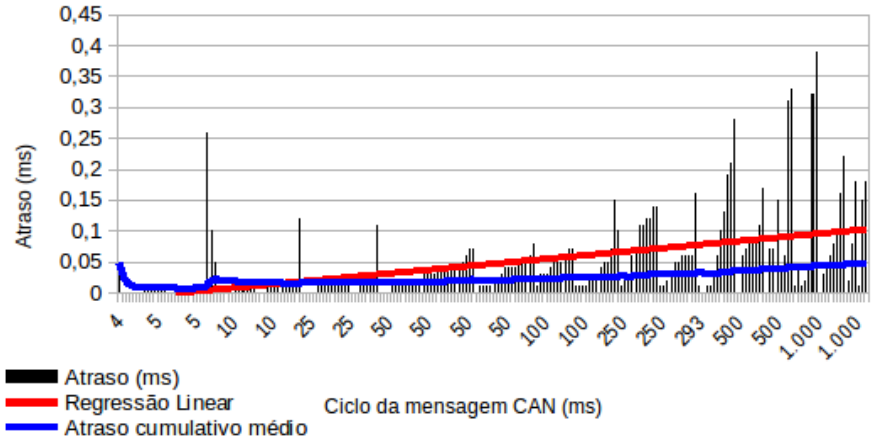


Fonte: Autor

Analisando o comportamento desta estratégia em relação ao Ciclo x Atraso (Figura 35) nota-se que as mensagens que estão nos extremos do eixo X (menores e maiores ciclos) obtiveram atrasos muito pequenos e as mensagens que ficaram distribuídas mais no meio do eixo X (ciclos medianos) obtiveram um atraso maior. As mensagens com ciclos pequenos sofrem pouco atraso por dispararem o gatilho de envio mais facilmente e exatamente pelo fato do gatilho ser disparado com mais frequência e as mensagens de

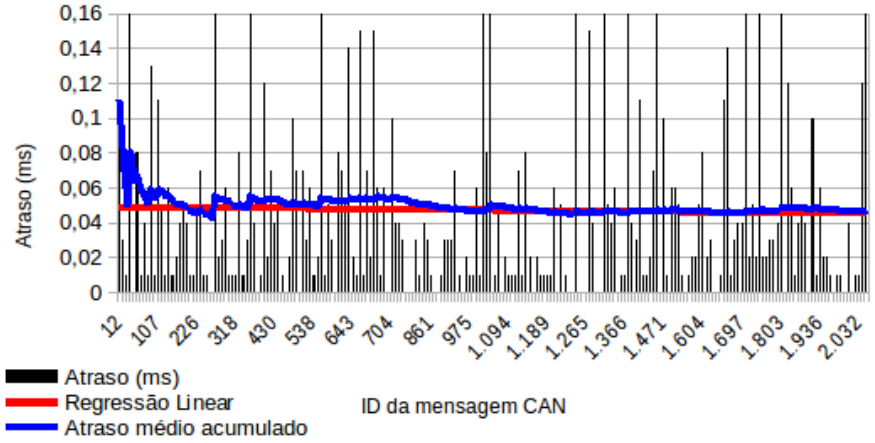
ciclo maiores ocorrerem com menor incidência no barramento, elas também acabam por sofrer menor atraso.

Figura 35 – Gatilho Estocástico por Ciclo (Ciclo x Atraso)



Fonte: Autor

Figura 36 – Gatilho Estocástico por Ciclo (ID x Atraso)



Fonte: Autor

Conforme ilustra a Figura 36, considerando o atraso médio por CAN ID a estratégia de Gatilho Estocástico por Ciclo apresentou um maior atraso médio por ID de 0,39 ms, entretanto com 50% dos CAN IDs sofrendo um atraso médio abaixo de 0,02 ms. Ainda, vale ressaltar que o algoritmo proposto conseguiu realizar o enfileiramento e multiplexação de CANs dentro de Ethernet com um atraso médio por ID de 0,05 ms (e desvio padrão de 0,06 ms) e, ao comparar Figura 36 com a Figura 17, pode-se notar que a estratégia de multiplexação Um-para-um apresentou um atraso médio por ID de 0,03 ms (e desvio padrão de 0,05 ms). Comparando estatisticamente estas duas estratégias, 99% dos CAN IDs sofreram um atraso médio de até 0,33 ms e 0,25 ms, respectivamente.

4.2 Comparativo Geral das Estratégias

A proposta deste trabalho de conclusão de curso, um algoritmo para enfileiramento e multiplexação de mensagens CAN em pacotes Ethernet, apresentou no cenário de 100% de carga resultados compatíveis com a estratégia Um-para-um (KERN et al., 2011), de baixíssimo atraso. Entretanto, a proposta deste trabalho traz como diferencial uma utilização do canal Ethernet 33% maior que a estratégia Um-para-um (KERN et al., 2011), o que equivale afirmar que o enfileiramento do algoritmo proposto é 77% mais eficiente do que apenas multiplexar uma mensagem CAN em um pacote Ethernet. Tal benefício tem o custo de, em média, 338 μs de atraso no despacho das mensagens CAN, para o cenário de 100% de carga simulado.

A Tabela 1 apresenta um comparativo entre as estratégias de enfileiramento e multiplexação, para despacho das mensagens CAN em pacote(s) Ethernet. Observe que as estratégias que apresentam um baixo atraso (ms) tem um característico desperdício de uso do canal Ethernet, inversamente proporcionais. Por exemplo, a estratégia de *Gatilho por Timeout do Primeiro Quadro CAN* apresentou o dobro de desperdício do canal Ethernet que o *Gatilho Estocástico por Prioridade*, mas ao custo de um atraso 2 vezes maior que este.

Tal valor de uso do canal (também conhecido como eficiência de protocolo) é calculado como sendo a proporção entre o *payload* utilizado (mínimo de 46 bytes) pelas mensagens CAN nele despachadas e o consumo do protocolo Ethernet com seus preâmbulos, cabeçalho e checksum (8+14+4 = 46 bytes). Fórmula do uso do canal:

$$usodocanal = \left(1 - \frac{26}{max(payload, 46)}\right) * 100 \tag{4.1}$$

A estratégia de *Buffer Limitado (N-1)* (KERN et al., 2011), por aguardar encher o pacote Ethernet com a quantidade configurada de mensagens CAN que nele caibam, apresentou o menor desperdício no uso do canal (1,75%, o mesmo que dizer que utiliza 98,25% da capacidade do canal Ethernet), mas ao custo de um atraso médio 238 vezes maior caso não fosse realizado o enfileiramento (**Um-para-um**). Observe que a estratégia *Um-para-um* (KERN et al., 2011) apenas encapsula uma mensagem CAN em Ethernet, não sendo considerada de enfileiramento pois tem *buffer* de tamanho unitário.

Observe na Tabela 1 que a segunda estratégia com melhor uso do canal Ethernet é uma das propostas deste trabalho, *Gatilho Estocástico por Prioridade*, alcançando um uso efetivo de 97,64% do canal Ethernet, ao custo de um atraso 1,4 vezes menor que o melhor caso, que seria a utilização máxima do *payload* Ethernet. Também, observe que a estratégia *Gatilho Estocástico por Ciclo* também proposta neste trabalho, apresentou o menor atraso médio por CAN ID bem como garantiu o menor atraso máximo absoluto

Tabela 1 – Resultados dos testes com utilização de 100% do barramento CAN

	Média dos Atrasos (ms)	**Desvio Padrão**	**Coeficiente de Variação (CV)**	**Quadros CAN por pacote Ethernet**	**Desperdício do Ethernet[1] (%)**
Um-para-um	0,02	0,01	0,71	1	56,52
Estocástico Ciclo	0,34	0,64	1,90	7	23,21
Gatilho por Prioridade	0,64	0,97	1,51	14	11,60
Gatilho por Tempo	1,70	1,40	0,82	31	5,24
Gatilho por Crédito	2,31	1,66	0,71	46	3,53
Estocástico Prioridade	3,40	2,76	0,81	69	2,36
Buffer	4,76	2,96	0,62	93	1,75

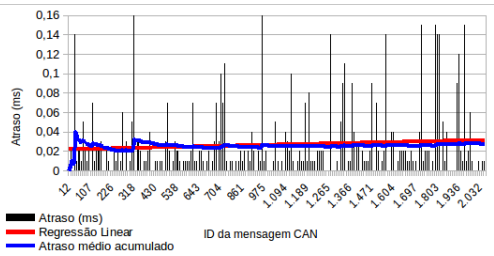
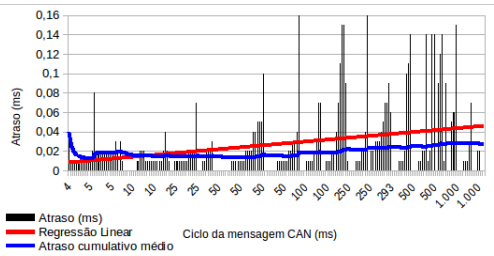
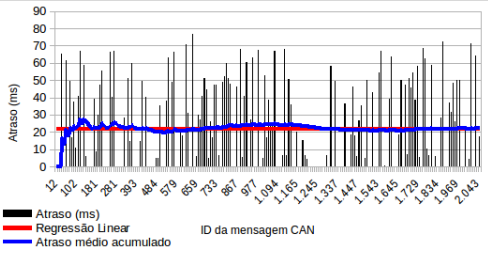
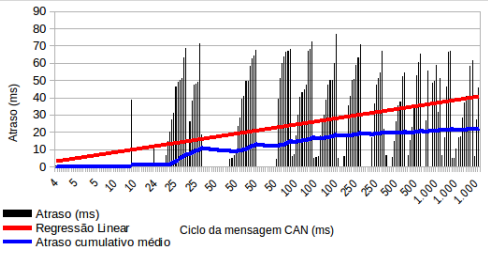
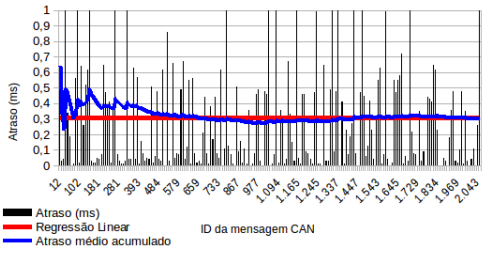
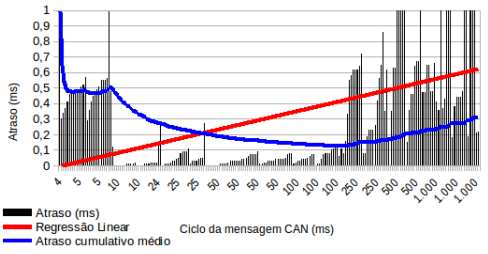
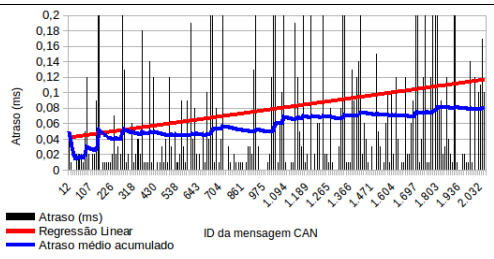
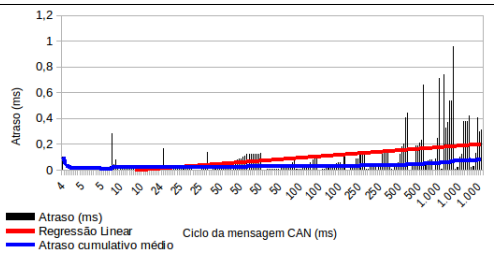
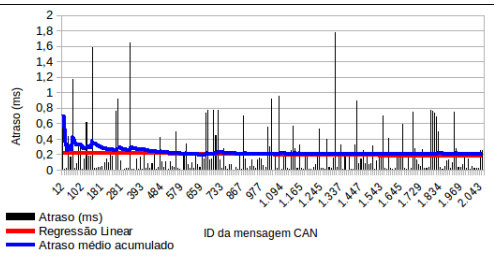
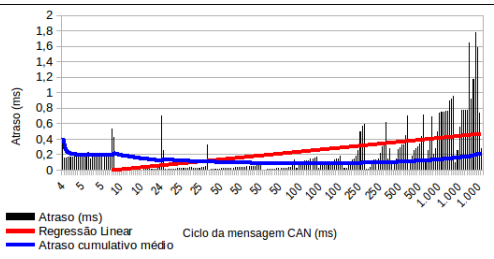
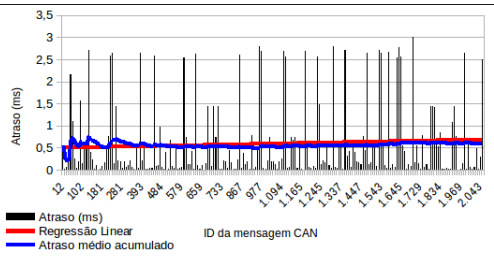
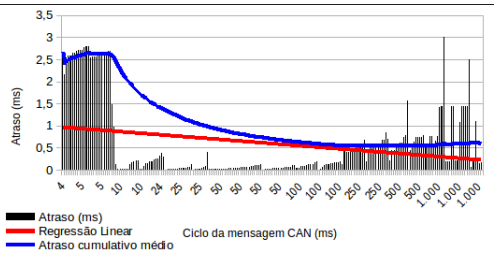
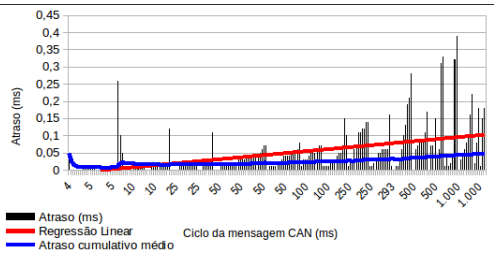
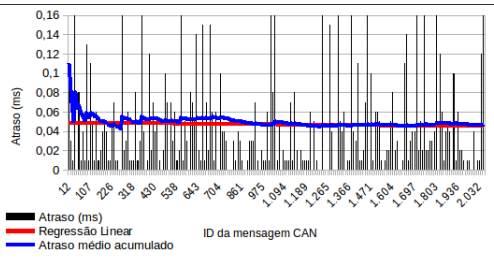
dentre todas as demais estratégias de enfileiramento. O algoritmo *Gatilho Estocástico por Ciclo* poderia ser utilizado por aplicações do barramento CAN que tolerem um atraso médio da ordem de 300 μs, ao passo que garantiu (no cenário com 100% de carga da CANbus) uma utilização de 76% do canal Ethernet, ou seja, 76 Mbps efetivos numa rede Ethernet de 100 Mbps nominais. Caso necessite-se de uma maior utilização efetiva do canal Ethernet, o algoritmo *Gatilho Estocástico por Prioridade* garantiu (no mesmo cenário) uma taxa efetiva de 97 Mbps numa rede Ethernet de 100 Mbps, só não sendo mais eficiente do que o melhor caso de forçar o despacho de uma fila sempre cheia, ao custo de um atraso médio por CAN ID de 3,4 ms e um atraso absoluto de no máximo 10,4 ms para 99% das mensagens CAN simuladas.

A Tabela 2 mostra lado a lado os gráficos de ID e Ciclo em relação ao atraso, atraso médio acumulado e regressão linear dos resultados obtidos para todas as implementações. Analisando o comportamento das estratégias desta maneira, é mais fácil perceber o comportamento de determinada estratégia em relação às outras. Os gráficos são apresentados na tabela na seguinte ordem:

- 1: Um-para-um;
- 2: *Buffer* Limitado;
- 3: Contador de Prioridade (Crédito)
- 4: Gatilho por Alta Prioridade;
- 5: Gatilho por *Timeout* do Primeiro Quadro;
- 6: Gatilho Estocástico por Pioridade;
- 7: Gatilho Estocástico por Ciclo.

Ao analisar o comportamento do ID x Atraso de todas as estratégias, percebemos que a estratégia de *Gatilho por Prioridade* (KERN et al., 2011) é a única estratégia com envio de pacote Ethernet baseado em prioridade de identificador em que a regressão linear mostra uma maior inclinação, enquanto nas outras estratégias a regressão apesar de crescente tende a ser mais suave. Vale notar que a estratégia de Gatilho Estocástico por Ciclo também apresentou um comportamento onde a regressão linear do atraso em função do ID apresentou uma maior inclinação. Já quando é observada a terceira coluna da Tabela 2 e são comparados os Ciclos x Atraso de todas as estratégias, o comportamento do *Contador de Prioridade (Crédito)* (KERN et al., 2011) e do *Gatilho Estocástico por Prioridade* se mostram semelhantes entre si e peculiar em relação às demais estratégias, pois a média do atraso acumulado começa alta para os quadros CAN com ciclos menores (no início do gráfico), cresce um pouco e depois diminui, enquanto no restante das estratégias esta média de atraso acumulado tende a se manter mais estável.

Tabela 2 – Resumo Gráfico do desempenho de todas as estratégias implementadas

#	ID x Atraso	Ciclo x Atraso
1		
2		
3		
4		
5		
6		
7		

5 Considerações Finais

O desenvolvimento deste projeto iniciou com o minucioso estudo dos fundamentos da rede CAN (e tecnologias *Fieldbus* em geral) e da família de protocolos Ethernet. Até onde conseguimos verificar, documentamos e comparamos as principais estratégias de interconexão dessas duas tecnologias, que tem como finalidade fornecer suporte à demanda crescente na vazão de dados para redes automobilísticas. Através do conhecimento obtido no referencial teórico levantado, foi possível implementar e comparar um total de sete abordagens para enfileiramento e multiplexação de quadros CAN em pacotes Ethernet. Quatro dessas implementações foram baseadas em trabalhos relacionados, três foram desenvolvidas a partir dos *insights* obtidos com a análise das limitações dos trabalhos relacionados. Todos os algoritmos implementados durante este trabalho foram testados através de simulação computacional, utilizando *softwares* disponibilizados pela indústria automotiva e tomando como caso de uso um cenário com 100% de carga de utilização do barramento CAN.

A maioria das implementações baseadas nos trabalhos de outros autores tiveram o comportamento esperado de acordo com suas descrições, fato importante como parâmetro de referência para as implementações inovadoras que foram realizadas neste trabalho. A estratégia de *Gatilho por Timeout do Primeiro Quadro* obteve um comportamento interessante no que diz respeito ao atraso do enfileiramento em relação ao ciclo da mensagem, proporcionando atrasos mais "apropriados" (proporcionais) para o tamanho dos ciclos de quadros CAN. Já o *Gatilho por Prioridade Estocástico* conseguiu utilizar o canal Ethernet de maneira mais efetiva. Por fim a estratégia de *Gatilho Estocástico por Ciclo* alcançou "o melhor dos dois mundos", obtendo atrasos mais baixos (perdendo apenas para uma ponte CAN-Ethernet sem *buffer* de enfileiramento).

A segunda estratégia com melhor uso do canal Ethernet é uma das propostas deste trabalho, *Gatilho Estocástico por Prioridade*, alcançando um uso efetivo de 97,64% do canal Ethernet, ao custo de um atraso 1,4 vezes menor que o melhor caso, que seria a utilização máxima do *payload* Ethernet. Já a estratégia *Gatilho Estocástico por Ciclo*, também proposta neste trabalho, apresentou o menor atraso médio por CAN ID bem como garantiu o menor atraso máximo absoluto dentre todas as demais estratégias de enfileiramento. O algoritmo *Gatilho Estocástico por Ciclo* poderia ser utilizado por aplicações do barramento CAN que tolerem um atraso médio da ordem de 300 μs, ao passo que garantiu (no cenário com 100% de carga da CANbus) uma utilização de 76% do canal Ethernet, ou seja, garantiria 76 Mbps efetivos numa rede Ethernet de 100 Mbps nominais. Caso necessite-se de uma maior utilização efetiva do canal Ethernet, o algoritmo *Gatilho Estocástico por Prioridade* garantiria (no mesmo cenário) uma taxa efetiva de 97 Mbps

numa rede Ethernet de 100 Mbps, só não sendo mais eficiente do que o melhor caso de forçar o despacho de uma fila sempre cheia, ao custo de um atraso médio por CAN ID de 3,4 ms e um atraso absoluto de no máximo 10,4 ms para 99% das mensagens CAN simuladas. Vale ressaltar que o algoritmo do *Gatilho Estocástico por Ciclo* também possui flexibilidade, pelo fato de seu comportamento poder ser alterado através do ajuste da condição de aceite para a ativação do gatilho, o que proporcionaria uma melhoria na vazão efetiva (diminuição da sobrecarga de protocolo) com pior atraso correspondente, ou vice-versa: redução do atraso porém com uma menor vazão efetiva.

O objetivo principal deste trabalho, de implementação e validação de pelo menos uma estratégia, foi alcançado bem como foram implementadas e validadas setes estratégias. Os principais resultados deste trabalho foram obtidos através das abordagens de *Gatilho Estocástico (por Prioridade e por Ciclo)* e pela abordagem de *Gatilho por Timeout do Primeiro Quadro*, que obtiveram resultados bons dentro dos parâmetros de referência de melhor e pior em atraso e vazão (cada estratégia lidou melhor com determinada característica do barramento) utilizados e abordaram uma visão diferente dos trabalhos relacionados específicos à estratégias de multiplexação, que podem ser interessantes em um cenário automobilístico.

As técnicas empregadas nas abordagens implementadas podem ser utilizadas conjuntamente à outras técnicas que não foram abordadas neste trabalho, como por exemplo o uso de múltiplas filas. É interessante prosseguir com esta pesquisa em trabalhos futuros ao utilizar gatilhos estocásticos ou gatilhos voltados aos ciclos dos quadros em ambientes de mais de uma fila, ou então utilizar mais de uma estratégia de enfileiramento em cenários de barramento que necessitem de maior granularidade na prioridade dos quadros do que apenas o valor do identificador CAN. Além da possibilidade de trabalhos futuros na continuidade da parte de *software*, também pode-se prosseguir com o estudo de utilização de outros equipamentos (como o Raspberry Pi) para obter o maior fluxo de processamento enquanto outros componentes distribuem as mensagens para os destinos corretos (como por exemplo um *switch*).

A principal relevância deste trabalho está na implementação e no estudo analítico do comportamento prático de abordagens de multiplexação que exploram diferentes técnicas e particularidades do barramento CAN (e de outras tecnologias) em comparação às estratégias mais comuns de enfileiramento. Através destas novas abordagens foi possível obter resultados melhores que os da literatura, cumprir todos os objetivos propostos para este projeto e abrir novos caminhos para experimentação da utilização da tecnologia Ethernet no cenário automobilístico.

Referências

ALLIANCE, O.; BROADR-REACH. Physical layer transceiver specification for automotive applications. IEEE Standards Association, 2014. Disponível em: <http://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.0.pdf>. Citado na página 36.

ARDUINO. *Arduino Mega 2560 Rev3*. 2017. Disponível em: <https://store.arduino.cc/usa/arduino-mega-2560-rev3>. Acesso em: 03/12/2017. Citado na página 45.

BUTTNER, J. D. Real-time ethernet: the ethercat solution computing and control engineering vol. 15. *Issue*, v. 1, p. 16–21, 2004. Citado na página 38.

CHEN, H.; TIAN, J. Research on the controller area network. In: IEEE. *Networking and Digital Society, 2009. ICNDS'09. International Conference on*. [S.l.], 2009. v. 2, p. 251–254. Citado na página 32.

CONSORTIUM, F. et al. Flexray communications system – protocol specification. v. 2, n. 1, p. 198–207, 2005. Disponível em: <https://goo.gl/o7JMxN>. Citado na página 30.

COOPERATION, M. Most media oriented system transportmultimedia and control networking technology. *MOST Specification Rev*, v. 2, 2004. Citado na página 30.

CORE, R. *Conversor de Nível Lógico RC - Robo Core Site*. 2017. Disponível em: <https://www.robocore.net/loja/produtos/conversor-de-nivel-logico.html>. Acesso em: 03/12/2017. Citado na página 56.

DX. *Arduino Pro Mini - DX Site*. 2017. Disponível em: <http://www.dx.com/p/arduino-pro-mini-microcontroller-circuit-board-blue-5v-16mhz-178183#.WfXE35-nFFQ>. Acesso em: 03/12/2017. Citado na página 46.

DX. *MCP2515 - DX Site*. 2017. Disponível em: <www.dx.com/p/434988>. Acesso em: 03/12/2017. Citado na página 47.

EMERSON. *Process Management – Fieldbus Communications*. SlideShare, 2017. Disponível em: <https://goo.gl/mnGMn1>. Acesso em: 03/12/2017. Citado na página 28.

FALLIS, A. *CAN System Engineering*. [s.n.], 2013. v. 53. 1689–1699 p. ISSN 1098-6596. ISBN 978-1-4471-5612-3. Disponível em: <http://link.springer.com/10.1007/978-1-4471-5613-0>. Citado 6 vezes nas páginas 30, 31, 32, 33, 34 e 35.

FILIPEFLOP. *Wiznet 5100 - Filipeflop Site*. 2017. Disponível em: <https://www.filipeflop.com/produto/ethernet-shield-w5100-para-arduino/>. Acesso em: 03/12/2017. Citado na página 46.

FOUNDATION, R. P. *Raspberry Pi Oficial Site*. 2017. Disponível em: <https://www.raspberrypi.org/>. Acesso em: 03/12/2017. Citado na página 44.

FREDRIKSSON, L.-B. A can kingdom (rev. 3.01). *Published by KVASER AB, Box 4076, S-51104 Kinnahult, Sweden*, 1996. Citado na página 21.

GOTHARD, D. et al. Tissue engineered bone using select growth factors: a comprehensive review of animal studies and clinical translation studies in man. *European Cells and Materials*, v. 28, p. 166–208, 2014. Citado na página 38.

HANK, P.; SUERMANN, T.; MÜLLER, S. Automotive ethernet, a holistic approach for a next generation in-vehicle networking standard. *Advanced Microsystems for Automotive Applications 2012*, Springer, p. 79–89, 2012. Citado na página 36.

HERBER, C. et al. Real-Time Capable CAN to AVB Ethernet Gateway Using Frame Aggregation and Scheduling. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, p. 61–66, 2015. ISSN 15301591. Citado na página 23.

HOGENMÜLLER, T. 1 Twisted Pair 100 Mbit/s Ethernet (1TPCE). In: *IEEE Standard for Local and Metropolitan Area Networks*. IEEE Standards Association, 2014. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado na página 36.

IEEE 802.1AS, S. IEEE Std 802.1AS-2011/Cor 1-2013 - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. In: IEEE (Institute of Electrical and Electronics Engineers), I. (Ed.). *IEEE Standards*. New York, NY, EUA: IEEE Standards Association, 2013. ISBN 9780738185873. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado 2 vezes nas páginas 22 e 38.

IEEE 802.1BA, S. IEEE Std 802.1BA-2011 - Audio Video Bridging (AVB) Systems IEEE. In: *IEEE Standards*. IEEE Standards Association, 2014. v. 2014, n. September, p. 74. ISBN 9780738192192. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado 2 vezes nas páginas 22 e 38.

IEEE 802.1Qat, S. 802.1qat-2010 - ieee standard for local and metropolitan area networks—virtual bridged local area networks amendment 14: Stream reservation protocol (srp). In: IEEE (Institute of Electrical and Electronics Engineers), I. (Ed.). *IEEE Standards*. New York, NY, EUA: IEEE Standards Association, 2010. ISBN 978-0-7381-6501-1. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado 2 vezes nas páginas 22 e 38.

IEEE 802.3, S. IEEE 802.3 Ethernet . In: *IEEE Standards*. IEEE Standards Association, 2009. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado na página 36.

IEEE 802.3, S. IEEE 802.3-2015 Standards. In: *IEEE Standards*. [S.l.]: IEEE Standards Association, 2015. Citado 3 vezes nas páginas 49, 50 e 53.

IEEE 802.3ab, S. 802.3ab-1999 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. Supplement to Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Physical Layer Parameters and Specifications for 1000 Mb/s Operation Over 4-Pair of Category 5 Balanced Copper Cabling, Type 1000BASE-T. In: *IEEE Standards*. [S.l.: s.n.], 2015. Citado na página 36.

IEEE 802.3bp, S. 802.3bp-2016 - ieee standard for ethernet amendment 4: Physical layer specifications and management parameters for 1 gb/s operation over a single twisted-pair copper cable. In: *IEEE Standards*. IEEE Standards Association, 2016. v. 2016. Disponível

em: <http://standards.ieee.org/about/get/index.html>. Citado 2 vezes nas páginas 22 e 30.

IEEE 802.3bw, S. IEEE Std 802.3bw-2015 - 100 Mb/s Operation over a Single Balanced Twisted Pair Cable, Physical Layer Specifications and Management Parameters for. In: *IEEE Standards*. IEEE Standards Association, 2015. v. 2015. ISBN 9781504401371. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado 4 vezes nas páginas 22, 30, 36 e 59.

IEEE 802.3u, S. Eee standards for local and metropolitan area networks: Supplement - media access control (mac) parameters, physical layer, medium attachment units, and repeater for 100mb/s operation, type 100base-t. In: *IEEE Standards*. IEEE Standards Association, 1995. Disponível em: <http://standards.ieee.org/findstds/standard/802.3u-1995.html>. Citado na página 59.

IEEE 802.Qav, S. Ieee std 802.1qav-2009 - ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams. In: *IEEE Standards*. IEEE Standards Association, 2009. v. 2009. ISBN 978-0-7381-6144-0. Disponível em: <http://standards.ieee.org/about/get/index.html>. Citado 2 vezes nas páginas 22 e 38.

ISO, I. Iso/iec 9899: 1999 - programming languages – c. *ISO/IEC JTC 1/SC 22 Programming languages, their environments and system software interfaces*, 1999. Citado na página 43.

ISO, I. 11898-1: 2003 - road vehicles – controller area network. *International Organization for Standardization, Geneva, Switzerland*, 2003. Citado na página 29.

ISO, I. Iso 11898-4: 2004 road vehicles – controller area network (can) – part 4. *International Organization for Standardization, Geneva, Switzerland*, 2004. Citado na página 29.

ISO, I. Iso 11898-3: 2006 road vehicles – controller area network (can) – part 3. *International Organization for Standardization, Geneva, Switzerland*, 2006. Citado na página 29.

ISO, I. Iso 11898-5: 2007 road vehicles controller area network (can) part 5. *International Organization for Standardization, Geneva, Switzerland*, 2007. Citado na página 29.

ISO, I. Iso 11898-6: 2013 road vehicles controller area network (can) part 6. *International Organization for Standardization, Geneva, Switzerland*, 2013. Citado na página 29.

ISO, I. Road vehicles – flexray communications system – part 1: General information and use case definition. *International Organization for Standardization*, p. 24, 2013. Citado na página 30.

ISO, I. Iso 11898-2: 2016 road vehicles – controller area network (can) – part 2. *International Organization for Standardization, Geneva, Switzerland*, 2016. Citado na página 29.

IXIA. *Automotive Ethernet*: An overview. [S.l.], 2014. 1–20 p. Disponível em: <https://goo.gl/VWSY9q>. Citado na página 23.

JOHANSSON, K. H.; TÖRNGREN, M.; NIELSEN, L. Vehicle applications of controller area network. *Handbook of networked and embedded control systems*, Springer, p. 741–765, 2005. Citado na página 29.

JUNGER, M. *Introduction to J1939*. VECTOR, 2010. Disponível em: <https://goo.gl/apZBgD>. Citado na página 29.

JUNIOR, H. T. Estudo dos protocolos de comunicação das arquiteturas eletroeletrônicas automotivas, com foco nas suas características e respectivas aplicações, visando o direcionamento para o uso adequado e customizado em cada categoria de veículo. *Centro Universitário do Instituto Mauá*, 2012. Citado na página 22.

KERN, A. *Ethernet and IP for Automotive E/E-Architectures-Technology Analysis, Migration Concepts and Infrastructure*. 219 p. Tese (Doutorado) — Universidade de Erlangen-nuremberg, Erlangen, 2012. Citado 4 vezes nas páginas 7, 9, 22 e 36.

KERN, A. et al. Gateway strategies for embedding of automotive CAN-frames into Ethernet-packets and vice versa. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6566 LNCS, p. 259–270, 2011. ISSN 03029743. Citado 7 vezes nas páginas 39, 50, 66, 68, 70, 78 e 80.

KOPETZ, H.; GRUNSTEIDL, G. Ttp-a time-triggered protocol for fault-tolerant real-time systems. In: IEEE. *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*. [S.l.], 1993. p. 524–533. Citado na página 38.

LEE, Y.; PARK, K. Meeting the real-time constraints with standard ethernet in an in-vehicle network. In: IEEE. *Intelligent Vehicles Symposium (IV), 2013 IEEE*. [S.l.], 2013. p. 1313–1318. Citado na página 40.

LOUREIRO, M. *Mais concorrência, menos carro 1.0 básico*. 2013. Disponível em: <https://exame.abril.com.br/revista-exame/adeus-carroca/>. Acesso em: 01 nov. 2017. Citado na página 21.

NACER, A. A. et al. Strategies for the interconnection of can buses through an ethernet switch. In: IEEE. *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*. [S.l.], 2013. p. 77–80. Citado na página 40.

NATALE, M. D. et al. *Understanding and using the controller area network communication protocol*: theory and practice. [S.l.]: Springer Science & Business Media, 2012. Citado na página 29.

OTHMAN, H. et al. Controller area networks: Evolution and applications. In: IEEE. *Information and Communication Technologies, 2006. ICTTA'06. 2nd*. [S.l.], 2006. v. 2, p. 3088–3093. Citado na página 30.

PETREVSKI, D. *Ligação entre Arduino UNO e MCP2515*. 2017. Disponível em: <http://blog.dimitarmk.com/2017/02/05/gmlan-sniffing-arduino-mcp2515/>. Acesso em: 03/12/2017. Citado na página 48.

SAUERWALD, M. Can bus ethernet or fpd-link: Which is best for automotive communications. *Analog Appl. J*, 2014. Citado na página 36.

SCHNEELE, P. Avionics full duplex ethernet and the time sensitive networking standard. *IEEE 802.1 Interim Meeting*, IEEE, 2015. Citado na página 38.

SEDGWICK, D. *Cars become computers on wheels*. 2014. Disponível em: <http://www.autonews.com/article/20140421/OEM06/304219993/cars-become-computers-on-wheels>. Acesso em: 01 nov. 2017. Citado na página 21.

SOUZA, A. G. de; NERY, F. H.; CAMPOS, G. L. *Estudo De Alternativas à Rede CAN Automotiva*. [S.l.], 2016. Citado 2 vezes nas páginas 21 e 22.

THIEL, C.; KÖNIG, R. Media oriented systems transport (most [r]) standard for multimedia networking in vehicle environment. *VDI BERICHTE*, VDI VERLAG GMBH, v. 1415, p. 819–834, 1998. Citado 2 vezes nas páginas 30 e 38.

THOMESSE, J.-P. Fieldbus Technology in Industrial Automation. *Proceedings of the IEEE*, v. 93, n. 6, p. 1073–1101, 2005. ISSN 0018-9219. Disponível em: <http://ieeexplore.ieee.org/document/1435740/?arnumber=1435740>. Citado 2 vezes nas páginas 27 e 28.

TINDELL, K.; BURNS, A.; WELLINGS, A. J. Calculating controller area network (can) message response times. *Control Engineering Practice*, Elsevier, v. 3, n. 8, p. 1163–1169, 1995. Citado 2 vezes nas páginas 35 e 41.

TOVAR, E.; VASQUES, F. Real-time fieldbus communications using profibus networks. *IEEE transactions on Industrial Electronics*, IEEE, v. 46, n. 6, p. 1241–1251, 1999. Citado na página 30.

TUOHY, S. et al. Intra-vehicle networks: A review. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 16, n. 2, p. 534–545, 2015. Citado na página 30.

VECTOR. *Ethernet - CANoe And CANalyzer*. [S.l.]: VECTOR, 2016. Citado na página 30.

YUNA, Y.; XIONG, H.-g. A method for bounding afdx frame delays by network calculus [j]. *Electronics Optics & Control*, v. 9, p. 014, 2008. Citado na página 38.