UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FERNANDO HENRIQUE ATAIDE

PROPOSTA DE MELHORIA DE TEMPO DE RESPOSTA PARA O PROTOCOLO FTT-CAN - ESTUDO DE CASO EM APLICAÇÃO AUTOMOTIVA

Porto Alegre
2010

FERNANDO HENRIQUE ATAIDE

PROPOSTA DE MELHORIA DE TEMPO DE RESPOSTA PARA O PROTOCOLO FTT-CAN - ESTUDO DE CASO EM APLICAÇÃO AUTOMOTIVA

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre
2010

FERNANDO HENRIQUE ATAIDE

PROPOSTA DE MELHORIA DE TEMPO DE RESPOSTA PARA O PROTOCOLO FTT-CAN - ESTUDO DE CASO EM APLICAÇÃO AUTOMOTIVA

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador:_____
Prof. Dr. Carlos Eduardo Pereira, UFRGS
Doutor pela Technische Universitat Stuttgart, Alemanha

Banca Examinadora:

Prof. Dr. Joao Netto, UFRGS
Doutor em Ciências Aplicadas pela Université Catholique de Louvain, Bélgica

Prof. Dr. Marcelo Goetz, UFGRS
Doutor em Informática pela Universität Paderborn, Alemanha

Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor em Informática pelo INPG, Grenoble, França.

Coordenador do PPGEE:_____
Prof. Dr. Alexandre Sanfelice Bazanella

Porto Alegre, dezembro de 2010.

RESUMO

Nos últimos anos os sistemas embarcados tem-se tornado notório nos mercados de eletroeletrônicos de consumo, automação industrial e comercial e em veículos em geral. Grande parte destas aplicações possui restrições temporais, sendo assim caracterizadas como sistemas de tempo real embarcado. Atualmente, a computação distribuída tem alcançado este tipo de sistema e por razão principal em custos desses sistemas, alguns barramentos ou redes de comunicação vêm sendo empregados como plataforma de conexão entre módulos eletrônicos. Um exemplo de aplicação de sistemas embarcados distribuídos e de tempo real é a eletrônica embarcada em veículos automotores, onde se encontram várias unidades de controle eletrônico espalhadas interior desses veículos com diferentes funções e se comunicando via rede de comunicação. Algumas pesquisas importantes nesta área já apresentaram diferentes abordagens em sistemas distribuídos de tempo real (SDTR) objetivando cobrir a crescente demanda de desempenho, previsibilidade e confiabilidade dessas aplicações emergentes. Tais requisitos envolvem baixa latência de transmissão, baixa variabilidade no tempo (*jitter*), tolerância a falhas e suporte para atualizações futuras - flexibilidade. Particularmente na área automotiva, onde é considerada a possibilidade de substituição de dispositivosmecânicos/hidráulicos por sistemas eletrônicos, conhecidos como *"by-wire" systems*. Assegurar um comportamento previsível e confiável desses sistemas assim como agregar um nível de flexibilidade são características necessárias em grande parte de aplicações de SDTR. O modelo de comunicação FTT (*Flexible Time-Triggered*) apresentado nesta dissertação, apresenta um alto grau de flexibilidade em relação a outros protocolos, tais como TTCAN, TTP e FlexRay. Um sistema distribuído de tempo real baseado no modelo FTT se adapta às mudanças de requisitos da aplicação em tempo de execução, sendo possível adicionar novas unidades de controle eletrônico sobre a rede após a fase de projeto. Esta característica advém do escalonador dinâmico deste modelo de comunicação. Este trabalho apresenta algumas propostas de melhoria de desempenho de tempo de resposta do protocolo FTT-CAN, descrevendo alguns pontos negligenciados na atual especificação do protocolo. As propostas têm como foco a estratégia de disparo de mensagens e tarefas, sendo a primeira relacionada à transmissão de mensagens síncrona (ou *time-triggered*), onde existem dois inconvenientes que geram *jitter* neste segmento de transmissão; a segunda é relacionado ao disparo de tarefas, onde existem algumas deficiências na liberação de tarefas síncronas na atual especificação do protocolo FTT-CAN.

Palavras-chave: Sistemas de Tempo-Real, Escalonamento de Tempo-Real, FTT-CAN, Sistema Operacional de Tempo-Real, Linux.

ABSTRACT

Embedded computing systems have become widely used in many areas. The greater part of those systems has time constraints and therefore they can be characterized as real time embedded systems. Nowadays, distributed computing has reached the embedded application, where some fieldbuses are already being used as communication platforms.
Some important researches has presented different approaches in the real time distributed embedded system domain aiming to cover the growing demands of performance, predictability and reliability of emerging applications. Such requirements involve low latency, reduced jitter, time composability, fault-tolerance and support for future extensions – flexibility. Particularly in the automotive area, on which several mechanical and/or hydraulic systems are being replaced by electronic "by-wire"systems, the importance of ensuring predictable behavior while also presenting some degree of flexibility plays a key role. Regarding to the flexibility, the Flexible Time Triggered communication model stands out against the others ones due to its high degree of flexibility. In this context, the FTT communication model appears as an interesting approach due to its high degree of flexibility while still ensuring a deterministic timing behavior. A distributed system based on a FTT communication infrastructure can adapts to changing application requirements, making possible the addition of new messages and nodes during operation. In this way, the communication infrastructure needs to schedule newest messages on-line. This master's work presents some proposals to improve the FTT-CAN response-time and indicating some drawbacks in already presented approaches. The improvements are concerning messages and tasks scheduling. Despite of its interesting characteristics, FTT CAN present some negative aspects regarding its timing behavior: the issue is on the synchronous message transmission, where there are two neglected points that generate jitter in this traffic; the other one is tasks dispatching, where there are some deficiencies concerning synchronous tasks execution. These disadvantages were not discussed in literature yet. This work presents new proposals to task and message scheduling of FFT-CAN based applications, therefore overcoming some of the main drawbacks of the protocol.

Keywords: Real-Time System, Real-Time Scheduling, FTT-CAN, Real-Time Operating System, Linux.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACK	Acknowledgement
AMS	Asynchronous Messaging System
BG	Bus Guardian
CAN	Controller Area Network
CC	Controlador de comunicação
CNI	Communication Network Interface
CRC	Cyclic redundancy check
CSMA	Collision Sense Multiple Access
CSMA-CD	Collision Sense Multiple Access - Collision Avoidance
D2B	Domestic Digital Bus
DLC	Data Length Code
EC	Elementary Cycle
ECU	Electronic Control Unit
ET	Event-Triggered
FP	Fixed Priority
FTDMA	Flexible Time Division Multiple Access
FTT	Flexible Time-Triggered Comunication
I-Frame	Initialization Frame
LIN	Local Interconnect Network
MEDL	Message Descriptor List
MMU	Memory Management Unit
MOST	Media Oriented Systems Transport
N-Frame	Normal Frame
RTAI	Real-Time Application Interface
RTOS	Real-Time Operating System
SAE	Society of Automotive Engineers

SC	Static Cyclic
SDTR	Sistema Distribuído de Tempo Real
SMS	Synchronous Messaging System
TDMA	Time Division Multiple Access
TT	Time-Triggered
TTA	Time-Triggered Architecture
TTCAN	Time-Triggered over Controller Area Network
TTP/A	Time-Triggered Protocol for SAE Class A
TTP/C	Time-Triggered Protocol for SAE Class C
UART	Universal asynchronous receiver/transmitter

1 INTRODUÇÃO

O mercado de sistemas embarcados tem passado por um grande crescimento nos últimos anos provocado pela redução dos custos dos semicondutores. Este fato possibilita custos competitivos no processo de desenvolvimento de hardware, que por conseqüência viabiliza o aumento dos requisitos funcionais dos sistemas embarcados provocando aumento da complexidade dos softwares e hardwares. A alta disponibilidade de semicondutores no mercado estimula projetos de sistemas cada vez mais complexos, exigindo maior poder de processamento, disponibilidade de memória e periféricos, tal como conversor analógico digital/digital analógico, módulos PWM (*pulse width module*) e interfaces de rede CAN, Bluetooth, e, dependendo da aplicação, Ethernet. Isto justifica o crescente "market share"dos microcontroladores de 32bit nas aplicações embarcadas. O autor em (SUNDARAMOORTHY, 2007) apresenta o *market share* dos microcontroladores de 8, 16 e 32bit indicando que o último superou o primeiro no ano de 2008 por razão de seu baixo custo.

No ponto de vista dos sistemas distribuídos de tempo real (SDTR), estes estão tornando largamente empregados em diversas áreas de aplicação, tal como controles de processos industriais, automação residencial e em sistemas de eletrônica embarcada de veículos automotores. Estas aplicações possuem restrições de previsibilidade nas dimensões de tempo e de valor, mesmo em situações de presença de falhas visto que tais aplicações podem estar intrinsecamente ligadas a integridades de pessoas ou custos de infra-estrutura. A indústria automobilística, por exemplo, possui interesse em produzir veículos populares com controle eletrônico de direção, onde dispositivos mecânicos e hidráulicos serão substituídos por atuadores e sensores elétricos. Este interesse tem foco na diminuição de peso e de consumo de combustível.

Os principais requisitos envolvidos em SDTR para este tipo de aplicação são: comportamento determinístico das execuções de tarefas e das transmissões de mensagem, baixa variabilidade temporal (*jitter*) mesmo em situações de carga de processamento/transmissão, suporte a técnicas de tolerância a falhas e também flexibilidade quanto a futuras adaptações devido a novos requisitos funcionais após a fase de projeto. Considerando que um SDTR é formado por um conjunto de unidades de processamento/controle eletrônico distribuído em um dado espaço, estas devem ter um protocolo de comunicação provendo serviços confiáveis e seguros para a camada de software superior, a aplicação propriamente dita.

Neste contexto, na área automotiva, alguns protocolos comerciais sustentam o cumprimento destes requisitos. O primeiro é o protocolo TTP/C (TTA-GROUP, 2003) e sua *Time-Triggered Architecture* (TTA) - um resultado de muitos anos do trabalho iniciado em 1979 na Universidade de Tecnologia de Viena com o projeto de MARS. Outro é o sistema de comunicação de FlexRay (CONSORTIUM, Copyright 2004) desenvolvido pelo

consórcio FlexRay, formado por um forte grupo de industrias da área automotiva.

Uma característica comum destes protocolos é seu comportamento estático de troca de mensagens no segmento de controle - faixa de tempo destinada às transmissões de mensagens que possuem restrições temporais -, constituídas por informações periódicas. O arbitramento no barramento de comunicação[1] é essencialmente baseado em TDMA, onde os tempos de transmissão de todas as mensagens devem ser conhecidos com antecedência. Neste caso cada estação ou nó (também denominada como ECU - *Electronic control Unit* ou Unidade de Controle Eletrônico) da rede de comunicação possui um instante, fixado em tempo de projeto, para a transmissão de cada mensagem periódica. Esta estratégia de comunicação é denominada *time-triggered*. O protocolo de FlexRay, em particular, combina a estratégia *time-triggered* e outra denominada *event-triggered*. Esta última é definida como um instante de tempo dinâmico variável, destinado às transmissões de mensagens baseada em eventos e utiliza a estratégia denominada FTDMA (*flexible time-division multiple-access*) como método de acesso ao meio físico de comunicação.

A união dos paradigmas *time-* e *event-triggered* fornecem um grau de flexibilidade ao protocolo FlexRay ao contrario do protocolo TTP/C que é puramente estático. Esta característica é importante para muitos sistemas de controle digitais modernos, principalmente quando o objeto controlado não é completamente conhecido na fase de projeto. Exigindo, assim, um sistema de controle que possa ser adaptável. Um SDTR com um alto grau de flexibilidade pode prover uma melhor utilização de recursos de hardware tal como: processamento e memória; e com uma capacidade de integração de novas funções em resposta a demandas de requisitos funcionais. Ou seja, adaptativo.

O autor em (ALMEIDA, 2003) apresenta uma avaliação detalhada sobre as principais vantagens de prover flexibilidade em um SDTR.

Outro protocolo, embora não comercial, é o FTT-CAN (ALMEIDA; PEDREIRAS; FONSECA, 2002) (*Flexible Time-Trigerred Communication on CAN*) que consiste em uma plataforma de comunicação concebida com foco em flexibilidade na troca de mensagens periódicas, sustentando ainda as garantias temporais. Este protocolo combina um segmento dinâmico de transmissão de mensagens periódicas e outro de mensagens baseadas em eventos. Estes segmentos são temporalmente isolados no acesso ao barramento de comunicação CAN. Ou seja, o protocolo FTT-CAN suporta ambos os paradigmas: *time-* e *event-triggered*. Em oposição aos protocolos TTP/C e FlexRay, que possuem um controle estático do instante de transmissão de mensagens periódicas, no FTT-CAN toda a carga de comunicação no segmento *time-triggered* pode ser dinamicamente controlada em tempo de execução.

1.1 Motivação

Em geral, um SDTR possui inúmeras tarefas (ou processos) em execução na unidade de processamento (processador ou micro controlador) e inúmeras trocas de mensagens com outras unidades de controle eletrônico (ECUs). As tarefas e mensagens podem ser caracterizadas em duas categorias: tarefas e mensagens ativadas por tempo, as quais são vinculadas a um relógio periódico; e tarefas e mensagens ativadas por eventos assíncronos de forma esporádica, por exemplo, alarmes ou processos de diagnostico. Nas literaturas existentes até a data dessa dissertação, como por exemplo, em (TTA-GROUP, 2003), tais categorias são denominadas como *time-triggered* e *event-triggered* respectivamente. Nestes sistemas, se faz necessário sustentar os tempos de resposta considerando execução

[1] método de controle de acesso ao meio físico de comunicação.

de tarefas e transmissão de mensagens em ambos os paradigmas. Evitando que mensagens ou tarefas de um dado processo afetem o cumprimento das restrições temporais de outras.

Alguns protocolos para aplicação em SDTR adotam algumas de técnicas para tratar este problema. Porém, eles privilegiam um ou outro paradigma. Por outro lado, SDTR com infra-estrutura de rede baseado no protocolo CAN nativo não possui tratamento para estes inconvenientes que conduzem a uma perda de tempo de resposta do sistema.

1.2 Objetivos

Esta dissertação de mestrado apresenta uma proposta de melhoria de tempo de resposta do protocolo FTT-CAN em ambos os paradigmas, *time-* e *event-triggered* provendo um isolamento. As abordagens apresentadas neste trabalho foram implementadas em um sistema operacional Linux desenvolvido para aplicações embarcadas com suporte para aplicações de tempo real através da extensão RTAI (*Real Time Application Interface*). A plataforma de hardware utilizada tem como base o micro controlador Freescale ColdFire. A validação do trabalho proposto teve como foco uma aplicação baseada em um sistema automotivo, onde uma arquitetura eletroeletrônica hipotética foi idealizada para fins de validação da proposta do protocolo FTT-CAN e seguindo as tendências da área automotiva.

1.3 Delimitações do Trabalho

Esta dissertação corresponde a parte final do curso de mestrado do Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul. A implementação do protocolo FTT-CAN deste trabalho não aborda o processo de escalonamento *on-line* de mensagens e tarefas que deve ser realizado no nó mestre da rede quando seguindo a proposta original do protocolo. Técnicas de tolerância a falhas já apresentadas em literaturas sobre o protocolo FTT-CAN também não foram consideradas. A razão da não abordagem destas características suportadas pelo o presente protocolo se deve ao foco direcionado a implementação da proposta sugerida neste trabalho. Todos estes pontos não abordados serão tratados nos trabalhos futuros.

1.4 Organização da Dissertação

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta uma revisão de conceitos necessários para a compreensão da presente dissertação. Inicia com uma breve discussão sobre paradigmas de comunicação e programação de sistemas de tempo-real, comparando as duas abordagens mais frequentes no desenvolvimento de sistemas SDTR (o uso de prioridades e a consideração de requisitos temporais como ciclos e *deadlines*). Ainda, este capítulo descreve os principais protocolos de comunicação aplicáveis em SDTR. Na sequência, o capitulo 3 apresenta o protocolo FTT-CAN, o qual é também abordado nos demais capítulos da dissertação. No capítulo 4 são descritas as propostas desta dissertação, as quais buscam sanar alguns dos problemas negligenciados na implementação do protocolo original FTT-CAN, tais como inversão de prioridade e *jitter* na transmissão de mensagens no segmento síncrono e bloqueio de execução de tarefas assíncronas causado pela liberação de tarefas síncronas. O capítulo 5 descreve a implementação e validação experimental das propostas aqui apresentadas usando um estudo de caso de aplicação automotiva. Finalmente no capítulo 6 são apresentadas as conclusões.

2 REVISÃO DE CONCEITOS

2.1 Time- e Event-Triggered

Nos últimos anos, vários autores da área apresentaram discussões sobre os paradigmas de comunicação *time-* e *event-triggered* em diferentes protocolos em aplicações SDTR (ALBERT, 2004), principalmente na área automotiva (KOPETZ, 1997). Várias observações foram apresentadas a favor de uma ou de outra abordagem considerando algumas características, tal como: *jitter* de mensagens e de tarefas, previsibilidade e flexibilidade. Estes modelos definem características particulares relacionadas ao comportamento de execução de tarefas do software de aplicação e também o comportamento da plataforma de comunicação.

Em um sistema baseado no modelo *event-triggered* (ET) todas as ações são ativadas por ocorrências de evento ou através de significativas mudanças de estado. Pode-se dizer que tais sistemas possuem respostas rápidas no ponto de vista do tempo de reação a estímulos externos. Ou seja, pode-se considerá-los como dinâmicos. Por outro lado, sistemas implementados seguindo o modelo *time-triggered* (TT) têm todas as ações ligadas a um relógio periódico. As ações possuem instantes de tempo pré definidos para sua execução. E, por sua vez, tais sistemas são caracterizados como sistemas estáticos. Com instantes predefinidos para execuções e com tempos de resposta bem conhecidos devido a inexistência de processos concorrentes que podem tomar o tempo de execução ou transmissão de uma tarefa ou mensagem.

Inicialmente, sistemas ET e TT eram considerados de forma independente (TTA-GROUP, 2003), assumindo que um sistema era completamente ET ou TT. Porém, o aumento da complexidade dos SDTR motivou a junção ou um trabalho mútuo entre ambos os paradigmas em um mesmo sistema que demande previsibilidade tanto nos eventos periódicos quanto naqueles esporádicos. Os crescentes requisitos funcionais dos atuais SDTR mostram que ambos os modelos, ET e TT devem coexistir e interagir em uma infra-estrutura de comunicação. Albert em (ALBERT, 2004) apresenta uma comparação entre os conceitos ET e TT no ponto de vista de sistema de controle via redes de comunicação.

2.1.1 Comunicação Event-Triggered e Time-Triggered

Alguns conceitos relacionados ao comportamento dos modelos ET e TT são apresentados nesta seção no ponto de vista das atividades de comunicação em uma arquitetura de comunicação baseada em um canal *broadcast*. Duas principais características de sistemas de comunicação *broadcast* são: uma mensagem enviada para o canal de comunicação é recebida por todos os nós conectados na rede; e somente um nó pode enviar uma mensagem por vez. Em SDTR estas características básicas de comunicação em rede é um

grande causador de atrasos de tempo de resposta. Diferentes estratégias de acesso ao canal de comunicação foram propostas para minimizar tal inconveniente, mas nem todas são adequadas para aplicações em SDTR. Tais estratégias são denominadas, em inglês, como *media access control* (MAC) ou controle de acesso ao meio. Conforme o modelo de referência OSI[1], a estratégia de acesso ao meio e identificada na camada de enlace de dados ou *data link layer*. Aqui são listadas algumas das principais estratégias:

1. *Collision Sense Multiple Access* (CSMA): É um método de acesso probabilístico, onde um nó que possui mensagens a serem transmitidas primeiramente verifica a ausência de transmissão de mensagem no canal compartilhado antes de iniciar sua própria transmissão. Desta forma, sendo possível o caso em que mais de um nó não identifiquem nenhuma atividade no canal no mesmo instante e iniciem a transmissão das mensagens causando colisões. Por essa razão, CSMA é combinado com outra estratégia para resolver esta inconveniência.

   (a) cada nó transmissor paralisa sua transmissão quando colisões são percebidas no canal de comunicação e inicia uma nova tentativa de transmissão mais tarde, após um tempo aleatório. O protocolo Ethernet, comuns no dia a dia de qualquer pessoa é um exemplo de protocolo baseado nesta estratégia.

   (b) CSMA/CA (*collision avoidance*) cada mensagem tem um identificador único usado em um processo de arbitragem. Este identificador determina a prioridade da mensagem durante o processo de competição ao acesso ao canal de comunicação. Um nó emissor paralisa sua transmissão quando detecta outra mensagem com um identificador e por consequência uma prioridade mais alta sendo enviada no canal. O protocolo de comunicação CAN utiliza esta estratégia como controle de acesso.

2. *Token passing*: uma "ficha"(*token*) é passada a cada nó da rede, e com posse desta ficha o nó tem permissão de acesso ao canal para realizar transmissão de mensagem. A ficha é um mecanismo de controle que dá autoridade ao sistema para comunicar ou usar os recursos. Uma vez que a comunicação é realizada, a ficha é passada para o próximo nó em uma maneira sequencial ou cíclica. O *Token-bus* é um protocolo que utiliza esta estratégia.

3. *Mini-slotting*: cada nó tem um tempo exclusivo para transmissão associado ao tempo relativo ao inicio de um ciclo de comunicação, denominado ciclo de barramento. Se o instante de tempo passou sem nenhuma atividade de comunicação no canal, o próximo instante de tempo será usado por outro nó da rede. ARINC 629, Byteflight, e FlexRay utilizam esta estratégia como controle de acesso.

4. *Time Division Multiple Access* (TDMA): cada nó tem um tempo pré atribuído para transmissão de mensagens. O método de acesso é cíclico e estático, e permite que cada nó envie mensagens periodicamente sem qualquer interferência de outros nós. TTP/C (TTA-GROUP, 2003) e FlexRay (CONSORTIUM, Copyright 2004) usam esta estratégia.

[1] O modelo de referência OSI é um conceito composto por sete camadas, cada uma especifica uma função particular da rede de comunicação. O modelo foi desenvolvido pela ISO (International Organization for Standardization) em 1984, e é considerado como modelo primário de comunicação entre computadores.

5. *Master Slave*: um nó pode enviar mensagens no canal de comunicação somente a pedido de outro nó, denominado nó mestre da rede. Assim, cada nó efetua transmissão somente após a requisição do mestre. O processo é cíclico respeitando os períodos de transmissão de cada mensagem. LIN (CONSORTIUM, 2006), TTP/A (KOPETZ, 1997) são exemplos de protocolos que utilizam esta estratégia.

Considerando uma plataforma de comunicação de tempo real baseada no paradigma ET onde todas as ações são ativadas pela ocorrência de um evento ou mudança de estado, esta deve possuir um mecanismo de arbitração para os nós competirem o acesso ao barramento. A desvantagem principal deste tipo de comunicação é o congestionamento no barramento que resulta em imprevisibilidade dos tempos de resposta das mensagens.

Nos sistema ET, somente o nó transmissor tem o conhecimento sobre o momento exato do envio da mensagem, isto é, cada nó somente tem sua visão do tempo físico. Por outro lado, uma vantagem destes sistemas é menor tempo de resposta a eventos externos assíncronos em baixa situação de carga de comunicação no barramento. Um exemplo de evento assíncrono são alarmes. A reação do sistema é rápida porque o instante de transmissão das mensagens não é controlado pela plataforma de comunicação, isto é, um nó pode tentar acesso ao barramento de acordo com a demanda de transmissão. Dependendo da estratégia de acesso ao barramento, os tempos de transmissão serão imprevisíveis devido as colisões em situações de carga moderada ou alta no barramento de comunicação. Em suma, SDTR baseado no paradigma ET são mais flexíveis e permite uma melhor utilização de recurso, neste caso o barramento de comunicação. A largura de banda total disponível no barramento é compartilhada entre os nodos, e automaticamente readapta quando novos nodos são removidos ou incluídos mesmo se a comunicação esteja em andamento. Esta característica esta intrinsecamente ligada ao método de acesso ao meio de comunicação. Entretanto, o tempo de resposta de transmissão de mensagens destes sistemas é deteriorado quando a carga de comunicação aumenta. Mesmo em situações de baixa carga as mensagens podem sofrer variações no tempo de transmissão, que é negativo em aplicações de controle em SDTR com rígidas restrições temporais. O protocolo CAN (CIA, 2001b) é reconhecido como padrão de protocolo ET determinístico.

Em outra direção, um sistema TT é caracterizado por seu comportamento determinístico no tempo, pois conta com a estratégia de acesso ao barramento através de divisão de tempo (TDMA). Esta estratégia concede uma boa previsibilidade no tempo de resposta de transmissão. Todos os tempos de transmissão de mensagens são definidos seguindo uma base de tempo global conhecida por todos os nós do sistema e definida em tempo de projeto. Sistemas TT que possui sincronização de relógio são capazes de oferecer serviços de comunicação de dados cíclicos com latências fixas com baixo *jitter*. A largura de banda é dividida em instantes de tempos dedicados, onde somente um nó pode executar uma transmissão de cada vez, por esta razão colisões nunca acontecem em operação normal do protocolo e do sistema. Um típico protocolo de comunicação baseado no TDMA é o TTP/C (TTA-GROUP, 2003).

A desvantagem mais abalizada neste tipo nos sistemas TT é a falta de flexibilidade. Para toda inclusão de nó ou mensagem na rede de comunicação que não foi previamente conhecida(o), todo o sistema deve ser reprogramado ou reprojetado. Uma propriedade interessante destes sistemas é que todo nó possui aproximadamente a mesma visão de tempo global e conhecem os instantes em que cada transmissão de mensagem ocorre, indiferentemente se a mensagem é do próprio nó ou de um adjacente. A detecção de erro é realizada no próprio protocolo de comunicação verificando se toda mensagem foi recebida dentro de sua janela de tempo esperado. Esta última vantagem é um dos meios de

identificação de erros em outros nós presentes da rede, como por exemplo, um nó ausente (sem transmissão de mensagem em seu especifico instante de tempo).

O FlexRay (CONSORTIUM, Copyright 2004) e FTT-CAN(ALMEIDA; PEDREIRAS; FONSECA, 2002) são exemplos de protocolos mistos. Ambos suportam os paradigmas ET e TT. O ciclo de comunicação do FlexRay é dividido em diferentes segmentos de tempo: um estático, um dinâmico opcional, uma janela de símbolo e outro destinado ao tempo de inatividade de rede. Todos os serviços de comunicação são manipulados dentro dos segmentos estáticos e dinâmicos. O segmento estático é usado para tráfego de TT enquanto o outro é compartilhado entre os nós para transmissões ET. Similarmente, o FTT-CAN divide seu ciclo de comunicação em dois segmentos: um ET e outra fase TT com suas respectivas transmissões. A principal característica do FTT-CAN é o tráfego TT dinâmico ativado por um nó mestre. Adiante detalhes do protocolo FTT-CAN serão apresentados. Por outro lado, em TTP/C não possui nenhum segmento de tempo que trate as transmissões ET. Deste modo, as mensagens ET são transmitidas ao longo do tempo dentro dos instantes de tempos não utilizados para transmissões TT. A Tabela 1 resume os assuntos apresentados acima.

paradigm	***Event-triggered***	***time-triggered***
reaction to external events	*fast*	*slow*
control traffic latencies	*variable*	*fixed*
channel bandwidth	*shared*	*dedicated*
design complexity	*low*	*average*

Figura 1: Comparação entre os paradigmas ET e TT

2.1.2 Tarefas Event-Triggered e Time-Triggered

Os paradigmas ET e TT no ponto de vista de execução de tarefas serão discutidos nesta sessão.

Na abordagem ET, a execução de tarefa é iniciada através de uma ocorrência de evento em momento arbitrário no tempo, da mesma forma na abordagem de transmissão de mensagem. Tipicamente, SDTR são compostos de várias tarefas que executam em modo concorrente, podendo essas execuções gerar conflitos de uso de recurso, por exemplo, processador e acesso a memória. Tais conflitos são resolvidos atribuindo prioridades aos processos ou tarefas do sistema. Isto só é possível com a existência de um *kernel*[2] multitarefa com políticas de escalonamento baseado em prioridades. O escalonador do *kernel* é responsável por administrar filas de tarefas baseado em prioridade, realizando a seleção

[2]É o componente central do sistema operacional, serve de ponte entre o software de aplicação e o processamento real de dados feito a nível de hardware. A principal atividade do *kernel* é gerenciar os recursos do sistema (a comunicação entre componentes de hardware e software)

da tarefa como maior prioridade da fila e colocando-a para execução. O escalonador também é responsável pela troca de contexto[3] entre as tarefas concorrentes.

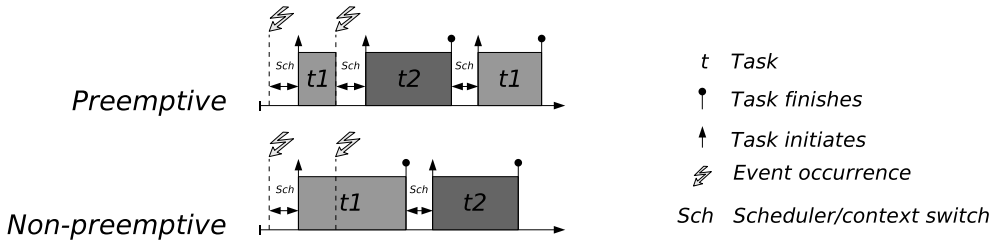

Figura 2: Execução de tarefas ET

A Figura 2 apresenta dois casos de execução de tarefas, cada um contendo duas tarefas ET concorrentes, $t1$ e $t2$. O primeiro considera um *kernel* com preempção e o segundo um *kernel* sem preempção. No primeiro caso, quando o segundo evento ocorre, a tarefa $t1$, que esta em execução, é imediatamente interrompida pelo escalonado do *kernel* devido sua prioridade mais baixa. Neste caso o escalonador efetua o disparo da tarefa mais prioritária, a tarefa $t2$. Então, o escalonador de tempo real colocará tarefa $t1$ em uma fila de tarefas e retomará sua execução após a conclusão da tarefa $t2$ e consequentemente efetuando a troca de contexto entre ambas as tarefas. Considerando o mesmo estudo de caso, agora com a condição de não preempção, a tarefa $t2$ aguardará até a conclusão da tarefa $t1$ mesmo tendo seu evento ocorrido durante a execução da última. E isso independentemente de sua prioridade. Como tarefa $t2$ tem a prioridade mais alta, a mesma será bloqueada em uma fila de tarefas até a próxima ativação do escalonador de tempo real. O paradigma ET possui vantagens no que tange sua flexibilidade e melhor uso dos recursos disponíveis. Embora apresente um consumo considerável de tempo de computação durante as ativações do escalonador de tempo real, porque um evento pode ocorrer durante o tempo de execução de uma dada tarefa. Sendo necessário efetuar várias trocas de contexto, no caso de *kernel* com preempção. Também a estas desvantagens acrescenta-se a complexidade das análises de escalonabilidade.

Da mesma forma que os sistemas de comunicação TT, a ativação de tarefa TT também ocorre em momentos predeterminados a priori. Então, os principais componentes do *kernel* para as ativações TT são: o gerenciador de interrupções e o *clock* do sistema. A exatidão de um sistema TT depende diretamente destes recursos. A ativação de tarefa pelo escalonador de tempo real é realizada através de uma tabela estática, contendo todas as ativações (Figura 3). A tabela estática armazena os instantes de liberação das respectivas tarefas. A Figura 3 apresenta três tarefas TT, cada tarefa com seu período T que serão ativadas de acordo com seus instantes de liberação. O período Ts - denominado de *master cycle* ou ciclo elementar - É o tempo em que todas as tarefas concluíram sua primeira execução antes de iniciar o próximo ciclo.

Um sistema TT baseado em escalonador cíclico com tabela estática tem baixa flexibilidade e apresenta uma baixa eficiência na utilização de recursos de sistema. Estas desvantagens o torna impróprio para aplicações em ambientes dinâmico. No entanto, sistemas TT têm uma propriedade atraente no ponto de vista do desenvolvimento do sistema, denominado *composability*, que é a capacidade de desenvolver dois subsistemas

[3]Troca de contexto ou chaveamento é o processo computacional de armazenar e restaurar o estado (contexto) de um processador de forma que múltiplas tarefas possam compartilhar uma única instância do processador. Garantido que quando o contexto anterior armazenado seja restaurado, o ponto de execução volte ao mesmo estado que foi deixado durante o armazenamento.

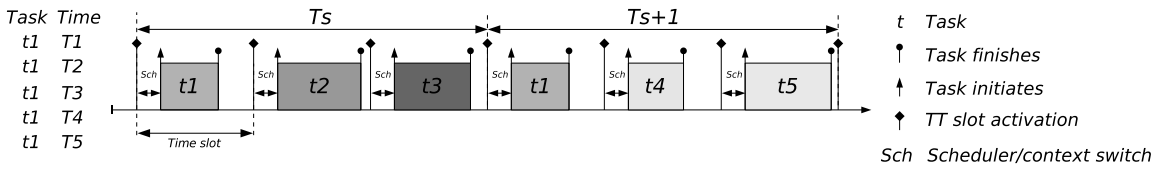

Figura 3: Execução de tarefas TT

independentes com garantias de que a integração de ambos não ocasionará perdas temporais e características funcionais que são individualmente definidas para cada componente (KOPETZ; BAUER, 2001).

Quando comparando complexidade de projeto, a implementação de sistemas TT é realmente mais complexa. Em cada nó do sistema deve-se assegurar que a camada de aplicação siga as execuções planejadas corretamente de forma síncrona. Isto significa que o tempo de execução no pior caso (WCET - *Worst Case Execution Time*) de todas as tarefas em um nó deve ser cuidadosamente estimado a fim de evitar inconsistências temporais. Mecanismos de sincronização são necessários nesses sistemas a fim de garantir as execuções e consequentemente a produção de saídas (resultados de uma execução de uma dada tarefa) consistentes e sincronizadas com os consumidores destas. Durante o tempo de projeto, não só o escalonamento da comunicação, mas também o escalonamento de todo as tarefas devem ser estaticamente definidos.

2.1.3 Coexistência de ambos os Paradigmas

Existem alguns protocolos que seguem ambos os paradigmas. Dois exemplos são FTTCAN (ALMEIDA; PEDREIRAS; FONSECA, 2002) e FlexRay(CONSORTIUM, Copyright 2004). Estes protocolos fazem uma união entre flexibilidade e determinismo pela combinação dos paradigmas ET e TT. Deste modo, é possível prover suporte para os requisitos dinâmicos para camada de aplicação como também garantir restrições temporais nas aplicações de tempo crítico.

Para adotar uma abordagem mista, o protocolo ou o *kernel* de tempo real deve prover mecanismo de isolamento temporal entre ambos os modelos a fim de evitar colisões entre execuções e ou transmissões de ambas os modelos. Este isolamento é necessário para manter todas as restrições de temporais. A estratégia comum usa um ciclo de comunicação dividida em duas partes distintas, um para ET e outro para tráfego de TT. Cada protocolo define sua própria denominação para ET e TT, por exemplo, nos protocolos FTT-CAN e de FlexRay são chamados de *time-* e *event-triggered phases* e *static/dynamic segments*, respectivamente.

Toda comunicação na parte TT é periódica e estática, exceto no protocolo FTT-CAN onde existe um nó mestre presente da rede que programa ou escalona mensagens nos demais nós que possuem controle de admissão de novas mensagens TT previamente não definidas. Tais recursos justificam o *"F"*, do *Flexible Time-Triggered*. Por outro lado, a comunicação ET é feita dinamicamente por acesso direto ao controlador de comunicação e, consequentemente, seguindo o processo de arbitragem nativo do protocolo de base, o CAN.

2.1.4 Reação a eventos assíncronos

A reação a eventos externos é uma característica importante em sistema SDTR, pois em determinadas aplicações o tempo de resposta a eventos assíncronos é um requisito de

sistema. O atraso fim a fim de um evento depende da previsibilidade do *kernel* do sistema operacional e escalonador de tempo real, e principalmente do determinismo do protocolo de comunicação.

Em um sistema completamente ET, por exemplo, baseado no protocolo CAN, a reação pode ser mais rápida do que em uma situação oposta, pois os instantes de transmissão não estão sob o controle da plataforma de comunicação como nos sistemas TT. Exemplificando no protocolo CAN, um evento externo acontece em um momento arbitrário no tempo em que o barramento de comunicação esteja ocupado com uma transmissão em andamento. Então, o nó respectivo esperará pelo próximo processo de arbitragem do protocolo CAN e neste instante, considerando o melhor caso, poderá ser capaz de efetuar transmissão em tal arbitramento. Basicamente, o tempo de resposta para evento externo no sistema CAN é influenciado por alguns fatores, tal como a carga de trabalho do barramento no momento exato da ocorrência do evento, a prioridade de mensagem do respectivo evento, comprimento máximo de mensagem presente no sistema e a taxa transferência usada na rede.

Em um sistema completamente TT a reação pode ser mais retardada dependendo do comprimento de segmento estático. Considere, por exemplo, que um evento ou requisição de transmissão de uma mensagem assíncrona chegue após a sua respectiva janela de transmissão. Neste caso, a transmissão estará atrasada pelo menos na duração inteira do ciclo elementar ou *master cycle*. A estrutura e o tempo do ciclo de comunicação, a taxa de transferência de dados ou latência de transmissão e a posição de ocorrência do evento no tempo são fatores importantes que impactam no tempo de resposta a eventos assíncronos nos sistemas TT.

2.2 Protocolos de comunicação de tempo-real para aplicações automotivas

Um SDTR é composto por um conjunto de nós interconectado que comunicam por um canal físico com algumas regras impostas pelo protocolo de comunicação. O sistema de comunicação é um dos componentes principais de um sistema de controle distribuído e, em alguns casos, é um componente crítico onde são requeridas técnicas de tolerância a falhas como, por exemplo, redundâncias. Com o crescente aumento de aplicações de segurança crítica nas novas gerações de sistemas embarcados automotivos como, por exemplo, aplicações *by-wire*, um comportamento determinístico nos serviços de comunicação é necessário. Tal exigência garante a previsibilidade da taxa de amostragem do processo de controle e do processo de atuação destas aplicações mesmo em altas condições de carga de trabalho no barramento de comunicação.

Os protocolos de comunicação adequados para sistemas automotivos de segurança critica são divididos em duas categorias já discutidas anteriormente de: protocolos de ET e TT. Uma variedade de protocolos já foi proposta para esta aplicação seguindo ambos ou um paradigma de comunicação. O protocolo CAN é o um dos mais antigos, mais conhecido e extensamente usado. Normalmente, existem duas redes CAN distintas nesse tipo de aplicação: uma com alta velocidade denominada *High Speed* CAN ou puramente C-CAN que é empregada em aplicações de controle motor, por exemplo; enquanto a outra de baixa velocidade denominada *Low Speed* CAN ou B-CAN é usada em aplicações de interior e conforto nos automóveis. A taxa máxima de transferência da rede CAN é 1 Mbit/s, mas ambas as redes operam em 500 e 50kbits/s ou 500 e 125kbits/s respectivamente. A baixa taxa de transferência de dados comparada com o potencial máximo do

CAN é escolhida a fim de garantir uma melhor imunidade à ruídos. A *Low Speed* CAN é tolerante a ausência de um dos canais do barramento CAN, sendo capaz de manter a comunicação confiável mesmo em um único canal. Isto é possível através da utilização de *transceivers*[4] específicos, como por exemplo o Philips 1054. O protocolo LIN (*Local Interconnect Network*) (CONSORTIUM, 2006) que foi desenvolvido nos últimos anos já é um protocolo difundido na área. Este tem como foco o baixo custo de infra-estrutura eletrônica, devido ser baseado em UART, com baixa taxa de transferência (20kbit/sec). O LIN é frequentemente usado em aplicações internas, geralmente discretas. Agora, no ponto de vista de informação e entretenimento onde se destaca a subárea denominada *infotainment*, existem os protocolos D2B (USA, 2004) e MOST (GRZEMBA, 2007) exclusivamente desenvolvido para aplicações multimídia. Atualmente, o protocolo MOST é o sucessor do D2B. Observando o gráfico da Figura 4, o FlexRay está entre o CAN e o MOST nos eixos de taxa de transferência e custo, porém é necessário notar as diferentes classes de aplicações automotivas apresentadas no lado direito da Figura 4.

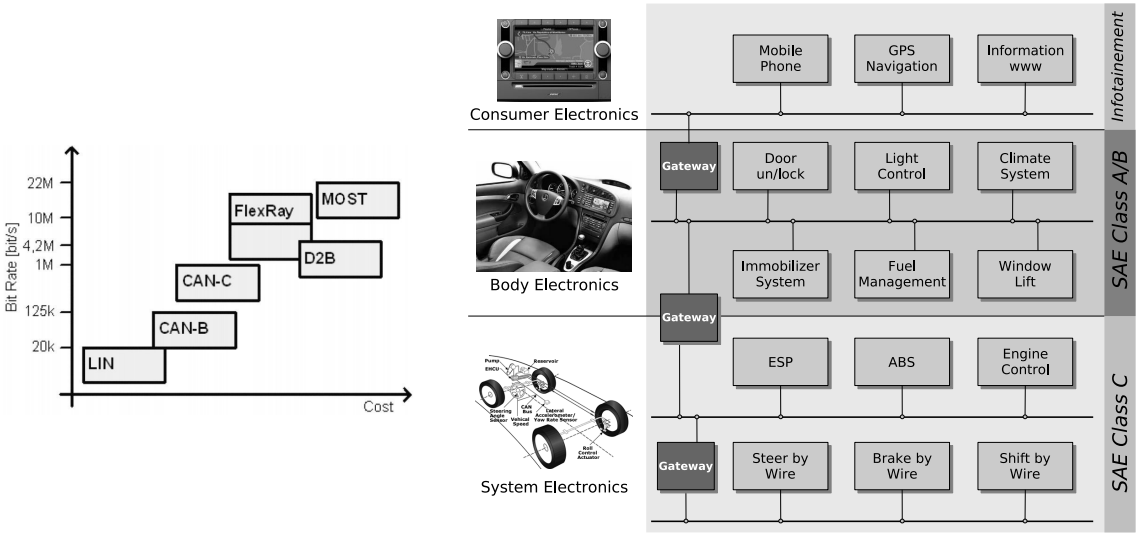

Figura 4: Custos versus taxa de transmissão de dados dos principais protocolos e suas aplicações

2.2.1 SAE Classification

Sobre as aplicações de redes de comunicação automotivas, a SAE (*Society of Automotive Engineers*) definiu três categorias básicas de aplicação de redes para SDTR em automóveis seguindo características relacionadas a determinismo temporal e requisitos de segurança. Abaixo é apresentado um resumo conceitual de cada categoria:

1. CLASSE A - Esta categoria compreende aplicações sem rígidas restrições temporais, ou seja, aplicação não tempo real. As exigências de largura de banda são baixas e esta categoria de redes automotivas não é empregada em sistemas de segurança critica. Exemplos de aplicações dessa categoria são: sistema de iluminação interna e externa, posicionamento e controle de acentos, espelhos retrovisores e vidros. Uma característica comum nestes exemplos de aplicações desta classe é sua carga útil de dados das mensagens e sua a taxa de atualização é relativamente

[4] *transceivers* é a combinação de transmissor e receptor em um mesmo dispositivo ou encapsulamento.

baixa. LIN (CONSORTIUM, 2006) e TTP/A (KOPETZ, 1997) são exemplos de protocolos desta categoria.

2. CLASSE B - Está presente em aplicações como sistema de controle automático de ar condicionado (por exemplo, sistema *dual-temp*), sistema de computador de bordo com informações de consumo instantâneo, médio, autonomia, tempo e distancia percorrida. Tais aplicações demandam uma largura de banda maior que a categoria anterior, isso devido às taxas de atualizações das variáveis envolvidas nestes sistemas. A Classe A e Classe B não são adequadas para aplicações de segurança crítica. O protocolo CAN é um grande exemplo desta categoria, e é largamente utilizado na indústria automotiva.

3. CLASSE C - Esta categoria inclui aplicações que são intrinsecamente envolvidas com a dinâmica do veículo, como, por exemplo, sistemas eletrônicos ativos de direção e freios e sistema automático de controle de câmbio. Nestes sistemas, a taxa de atualização de informações é na ordem de 1 a 10ms e a carga útil das mensagens transmitidas e recebidas nestes sistemas é maior se comparado às categorias anteriores, A e B. Além disso, aplicações da Classe C são ditas como de segurança crítica porque uma queda do sistema pode levar a consequências desastrosas e por este motivo estas demandam alta previsibilidade, confiabilidade e baixa latência. A fim de cumprir estes requisitos, as plataformas de comunicação com técnicas de tolerância a falhas são necessárias. Atualmente, TTP/C (TTA-GROUP, 2003) e FlexRay (CONSORTIUM, Copyright 2004) são exemplos de protocolos para aplicações Classe C.

Considerando os paradigmas apresentados anteriormente, o ET e TT, existem alguns protocolos presentes na indústria ou mesmo ainda em meio acadêmico que favorecem um ou outro, ou permite a coexistência de ambos. Nas subseções seguintes são apresentados alguns destes protocolos.

2.2.2 CAN

O CAN (*Controller Area Network*) é um protocolo de comunicação serial intrinsecamente *event-triggered*. Foi concebido e é mantido atualmente pela BOSCH, na Alemanha, para aplicações automotivas no início de 1980. Mas este protocolo não se restringe apenas para aplicações anteriormente citadas, atualmente existem variantes deste protocolo com foco em automação industrial. Exemplos destes protocolos são CANOpen (CIA, 2001a) e DeviceNet (CIA, 2001a). O CAN satisfaz as necessidades de comunicação em varias áreas, como rede de alta velocidade (1Mbps) e até pequenos sistemas multiplexados de baixo custo. Na indústria automotiva este protocolo é usado em unidades de controle de motor, quadro de instrumentos, e outros. Por outro lado, seu baixo custo de desenvolvimento de hardware e software possibilita seu emprego em componentes relacionado à interior e conforto veicular - iluminação interna, controle de vidros e bloqueio de portas - em substituição aos sistemas de chicotes tradicionais (CIA, 2001b). O CAN foi internacionalmente padronizado em 1993 como ISO 11898-1 e inclui a camada de enlace de dados do modelo de referencia de sete camadas ISO/OSI.

Além do CAN, foram desenvolvidos outros protocolos para o uso na indústria automotiva como o ABUS da Volkswagen, o VAN da Peugeot e Renault e o J1850 da Chrysler, General Motors e Ford. Estes protocolos diferem fundamentalmente ao nível da taxa de transferência, formato das mensagens, detecção de erros e seu tratamento.

O CAN é um barramento do tipo *broadcast*, isto quer dizer que todos os nós escutam todas as transmissões. Não existe forma de enviar uma mensagem para um determinado nó, pois todos os nós ouvem essa mensagem. O hardware do CAN, no entanto, pode conter filtros locais de forma que cada nó somente reaja a mensagens que têm interesse.

A transmissão de mensagens é realizada de forma assíncrona, somente através de requisições da aplicação, através de escrita direta nos *buffers* de mensagem e nos registradores do controlador CAN.

Cada mensagem deve ter seu próprio identificador que é usado para atribuir uma identificação do conteúdo da mensagem e também define a prioridade da mesma. O identificador com valor menor possui a prioridade mais alta. O protocolo de CAN define dois estados lógicos do canal de comunicação: dominante e recessiva. O estado dominante corresponde ao valor "0", e este sobrescreve o outro estado recessivo que por sua vez representa o valor lógico "1". A mensagem com maior quantidade de bits dominantes em seu campo identificador - que por outro lado é a que possui o menor valor - ganhará o processo de arbitramento e transmitirá seus bytes de dados. Um nó transmissor deve monitorar o canal para determinar se seu bit recessivo foi sobrescrito por um bit dominante de outro nó transmissor da mesma rede. Um nó que perde o processo de arbitramento imediatamente cessa sua transmissão e continua como receptor da mensagem em curso de transmissão.

O protocolo CAN possui quatro tipos de frames diferentes: frame de dados, frame remoto, frame de erro e um frame de sobrecarga.

Frame de dados Possui os seguintes agrupamento de bits:

- 1 bit de inicio de frame;
- campo identificador para identificação da mensagem e sua prioridade para o processo de arbitragem, possui 11 e 29 bits nas versões CAN 2.0A e 2.0B respectivamente;
- 1 bit RTR (remote transmission request) que identifica um frame remoto de requisição de mensagem;
- 6 bits de controle;
- 64 bits de dados;
- 16 bits de CRC;
- 2 bits de ACK(reconhecimento de mensagem recebida);
- 7 bits de fim de frame.

A estrutura de um frame de dados é apresentada na Figura 5.

Frame remoto Permite que um nó possa solicitar a transmissão de uma mensagem particular de outro nó, que tem o mesmo identificador do frame remoto transmitido. Isto também é usado para propósito de diagnóstico para checar se o nó produtor de uma dada mensagem está em operação normal.

Frame de sobrecarregue Este tipo de frame é constituído por 6 bits recessivos mais o delimitador. Este é enviado nas seguintes situações: quando um receptor não está preparado para aceitar uma mensagem e carece de mais tempo para se preparar; se detectado um nível dominante no campo de intervalo entre frames; ou se detectado um nível dominante no oitavo e último bit do delimitador de erro ou de sobrecarga. Este frame é tratado de forma diferente do frame de erro, pois o frame de sobrecarga não obriga à retransmissão do frame anterior.

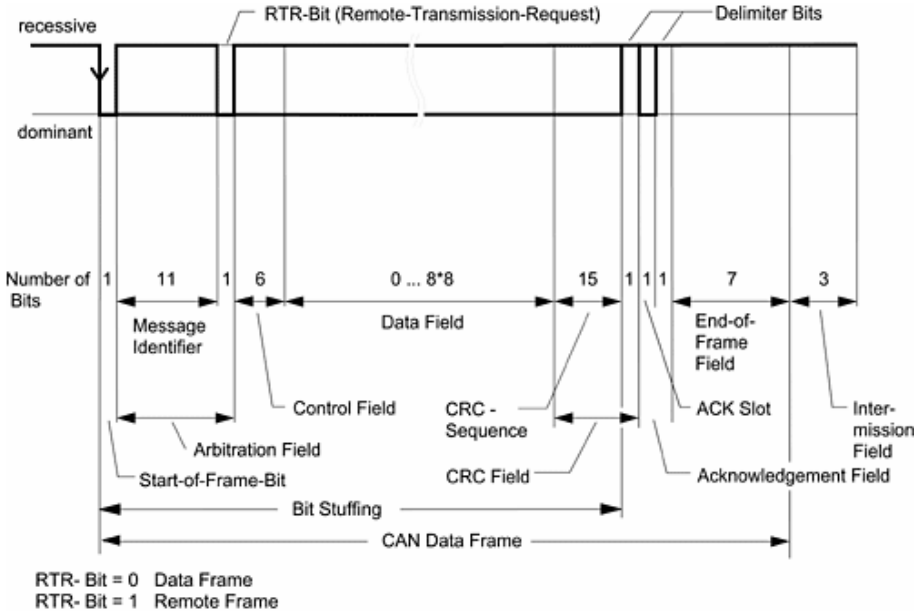

Figura 5: Frame de dados do protocolo CAN (CIA, 2001a)

Frame de erro O protocolo possui detecção de erros de modo que o nó transmissor possa retransmitir a mensagem corrigida. Qualquer nó presente na rede tenta encontrar erros na mensagem. Se for descoberto algum erro, o nó que o encontrou inicia a transmissão de um frame de erro, destruindo o tráfego existente na rede. Os outros nós, por sua vez, vão detectar o erro causado pelo frame de erro e tomam a ação correta, isto é, ignoram a mensagem corrente. Este frame possui 6 bits mais o delimitador, similar ao frame de sobrecarga. O frame de erro possui dois tipos de flags além do delimitador. O primeiro é o flag de erro ativo que consiste em 6 bits dominantes consecutivos e que é transmitido por um nó ativo para erro; o segundo é o flag de erro passivo, que consiste em 6 bits recessivos consecutivos. Este flag é transmitido por um nó passivo para erros.

Além dos frames citados acima, existe um espaço ente frames que é responsável por separa o frames do tipo dados ou remota de qualquer outro tipo que os precedem. Isto não acontece com os outros tipos de frames: erro e de sobrecarga.

O barramento CAN é classificado como par trançado diferencial. Este conceito atenua fortemente os efeitos causados por interferências eletromagnéticas, uma vez que qualquer ação sobre um dos fios será sentida também pelo outro, causando flutuação em ambos os sinais para o mesmo sentido e com a mesma intensidade. Como o que vale para os módulos que recebem as mensagens é a diferença de potencial entre os condutores CAN_H e CAN_L (e esta permanecerá inalterada), a comunicação não é prejudicada. Os níveis lógicos no barramento CAN, dominante e recessivo, são criados em função da condição presente nos fios CAN_H e CAN_L que compõe o meio físico da rede. A Figura 6 ilustra os níveis de tensão em uma rede CAN, assim como os bits dominantes e recessivos.

Na camada física, o protocolo usa codificação de NRZ (*Non Return to Zero*) com o método de *bit stuffing*. Na transmissão de um frame, a cada sequência de cinco bits idênticos um bit de diferente estado é inserido, exceto o campo CRC, campo de *acknowledge* e o de fim de frame não são codificados. Este método não é usado na transmissão de frames de erro e de sobrecarregar.

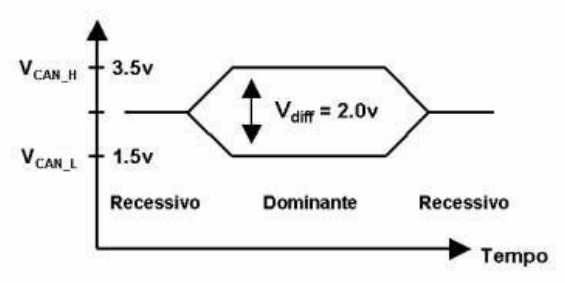

Figura 6: Características elétricas do barramento CAN (CIA, 2001a)

Cada nó CAN tem dois contadores de erros, sendo um contador de erros de transmissão e o outro contador de erros de recepção. Estes contadores podem assumir um dos três estados:

Erro Ativo o nó participa normalmente nos procedimentos da rede enviando um frame de erro ativa sempre que detecte um erro;

Erro Passivo neste estado o nó substitui o frame de erro ativo pela de erro passivo, passando a existir uma maior desconfiança que provavelmente esse erro é atribuível a ele próprio;

Bus Off ao entrar neste estado o nó retira-se de toda a atividade do barramento, ou seja, desativa os seus drivers de saída. Para regressar às atividades do meio deve existir uma reinicialização do sistema. Assim os seus contadores, TEC e REC, voltam a zero e o nó ao estado de erro ativo e à normal atividade no barramento.

Os dois contadores de erros são denominados como:

TEC (*Transmitter Error Counter*) ou contador de erros de transmissão;

REC (*Receiver Error Counter*) ou contador de erros de recepção.

Estes contadores são incrementados ou decrementados segundo determinadas regras. Os incrementos podem ser de 1 ou de 8 segundo a gravidade do erro. O decremento faz-se quando uma transmissão/recepção é bem sucedida, sendo o seu valor de uma unidade.

Devido a restrições físicas de hardware dos *transceiver* CAN, o limite prático de quantidade de nós em uma rede é de 110 nodos interligados em uma barramento. A taxa de transferência de dados depende do comprimento do barramento, sofrendo uma atenuação com o aumento do barramento, conforme ilustrado na Figura 7. Em aplicações industriais são utilizados repetidores.

2.2.3 LIN

LIN (Local Interconnect Network) é um protocolo mestre escravo *time-triggered*. Seu meio físico consiste de um único canal com taxa máxima de transmissão de 20Kbit/s. Sua aplicação principal está nos dispositivos discretos como, por exemplo, controle de acentos, sistema de travamento de portas, sensores de chuva e crepuscular chuva sensores, e luzes internas. Com baixa taxa de transmissão de dados, meio físico de um canal, o LIN possibilita um baixo custo de interconexão de sensores e atuadores quando há exigências que justificam uma rede CAN, por exemplo. Audi, BMW, DaimlerChrysler, Motorola,

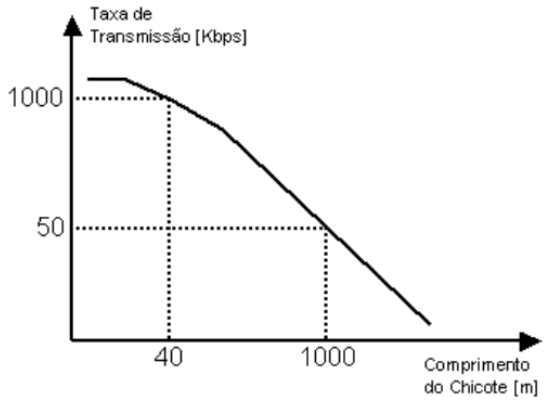

Figura 7: Atenuação da velocidade de transferência de dados com o aumento do tamanho do barramento (CIA, 2001a)

Vulcano, Volvo, e Volkswagen propuseram e desenvolveram este protocolo de padrão aberto de baixo custo. Seu meio físico é baseado no padrão UART/SCI que é comum nos microcontroladores atuais. A Figura 8 abaixo apresenta um simplificado diagrama de aplicação LIN.

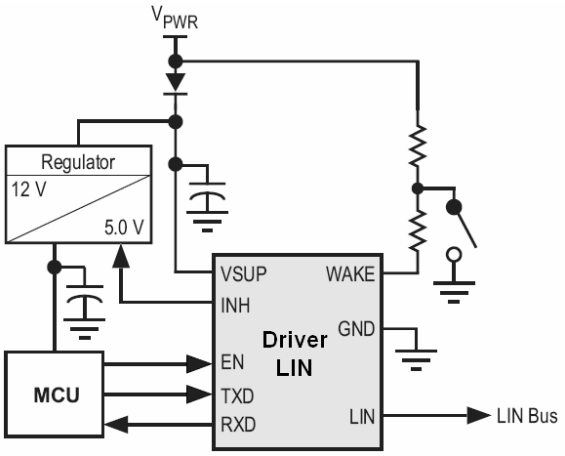

Figura 8: Componentes de hardware para uma rede LIN

LIN possui uma auto sincronização sem a necessidade cristal nos nós de escravo, deste modo provendo uma redução de custo significante de plataforma de hardware. Devido ao seu método de acesso ao meio baseado em mestre-escravo, o protocolo não necessita de arbitramento.

Um mestre pode controlar até 16 escravos em uma distância máxima de 40 metros, cabendo a ele o controle e o início do tráfego da mensagem no barramento. Com isso, o escravo não transmite até a autorização do mestre. Essa abordagem proporciona uma latência fixa da mensagem. Tais características reduzem a complexidade da implementação do sistema.

O protocolo LIN utiliza o conceito de tarefa. Em cada nó, todas as tarefas executadas, inclusive as tarefas de comunicação dos escravos, são alternadas entre tarefas de envio e de recepção, coordenadas pelo nó mestre. O formato frame é fixo, sendo composto por um cabeçalho (*header*) e um campo de resposta (*response*). O frame pode possuir entre zero e oito bytes de comprimento, seguidos de um byte de *checksum*.

2.2.4 TTCAN

Para superar a falta de previsibilidade e determinismo no protocolo CAN, Robert Bosch propôs uma extensão *time-triggered* para o protocolo. Denominada de TTCAN (RAHUL SHAH, 2002), (*Time-Triggered* CAN). O principal objetivo deste protocolo é evitar o *jitter* resultante da comunicação baseado em CAN nativo e garantir uma comunicação determinística para aplicações em SDTR.

No protocolo TTCAN a regra de comunicação segue uma mensagem de referência enviada por um nó mestre. Esta mensagem é a referencia de tempo global da rede, e baseada neste tempo algumas mensagens são atribuídas para transmissão. Uma estratégia semelhante também é empregada nos protocolos FTT (FTT-CAN e FTT-Ethernet) para obrigar uma referência de tempo para transmissão de mensagem, que será apresentada mais adiante. A especificação ISO 11898 foi estendida ganhando uma nova versão com suporte a comunicação TT que recebe o novo código ISSO 11898-4 em dois níveis: o primeiro garante a comunicação TT por meio da mensagem de referência de um nó mestre; no outro nível uma base de tempo globalmente sincronizado é fornecida e uma correção de desvio de tempo global entre os nós é estabelecida a fim de manter um sincronismo temporal.

A mensagem de referência tem um único identificador para ser reconhecida pelas nós presentes na rede. Esta possui um byte de informações de controle e os demais bytes podem ser usados para transferência dados. No segundo nível do protocolo, a mensagem de referência transporta uma informação de tempo globais do mestre. São 4 bytes para informações de controle e os demais bytes são usados para transferência de dados como nível 1 do protocolo.

O ciclo de comunicação - denominado ciclo básico - é o intervalo de tempo entre duas mensagens consecutivas de referência (Figura 9). O ciclo básico é composto de algumas janelas de tempo com tipos e tamanhos diferentes para transmissão de mensagens. A primeira janela, denominada de janela exclusiva, é usada para transmissões de mensagens TT. Uma outra janela de tempo, denominada janela de arbitragem, é preenchida com as transmissões ET, que concorrem através do mecanismo de arbitramento seguindo as prioridades de acordo com o valor presente no campo identificador nas mensagens seguindo o protocolo CAN nativo. Retransmissões automáticas usando RTR não são permitidas em ambas as janelas de tempo a fim de garantir a referência de tempo de cada janela, instante de inicio e fim. A última é a janela de tempo livre, que é um tempo reservado para extensões futuras da rede. O protocolo TTCAN permite alterar uma janela de tempo livre para uma de arbitragem ou exclusiva em caso de necessidades adicionais de largura de banda.

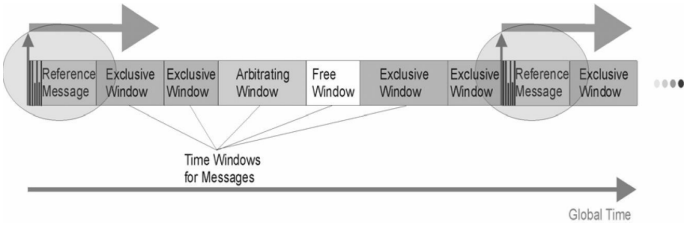

Figura 9: Ciclo de comunicação do protocolo TTCAN (RAHUL SHAH, 2002)

Como a tabela de mensagem é inteiramente definida de forma estática e em tempo de projeto, nenhum conflito acontece no sistema se a construção da tabela foi feita usando ferramentas de análise adequada. Em caso de qualquer distúrbio - devido a um nó de intruso - a arbitragem do protocolo CAN nativo é usado na ordenação das transmissões,

porem as mensagens envolvidas estarão sujeitas a *jitter* de acordo com a carga intrusiva presente.

2.2.5 TTP/C

O TTP/C é uma plataforma de comunicação baseado no TTP (*Time-Trigerred Protocol*) e desenvolvido com o objetivo de atender todos os requisitos das aplicações SAE classe C. Como citado anteriormente, essa classe de aplicação é categorizada como de segurança critica, exigindo determinismo e previsibilidade temporal além de tolerante a falhas. A origem do TTP/C foi no projeto MARS (*Maintainable Real-time System*), na Universidade Técnica de Vienna.

A estrutura do TTP/C é similar ao protocolo FlexRay. Cada nó consiste de um processador e um controlador de comunicação. Entre o processador e o controlador de comunicação existe uma interface de rede de comunicação (*communication network interface* - CNI). A CNI é a interface entre o controlador TTP/C e o processador dentro de uma ECU, basicamente é região de memória *dual-port* mapeada. É também denominada como uma interface de base de mensagens. Ela fornece ao processador uma área de memória para envio e recebimento de mensagens e informações de status do controlador de comunicação TTP/C. A Figura 10 apresenta os componentes da estrutura de uma ECU TTP/C.

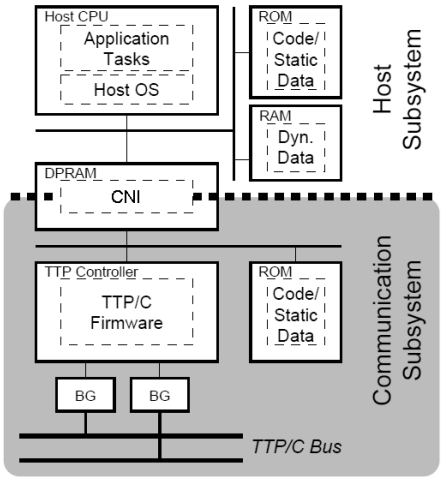

Figura 10: Estrutura de uma ECU TTP/C (TTA-GROUP, 2003)

O controlador TTP/C é suportado por dois guardiões de barramento (*bus-guardian*). Cada canal físico de comunicação é protegido por este dispositivo, os quais protegem o barramento de serem monopolizados por falhas em nós. Esta falha é conhecida como *babbling idiot failure*.

Na *Time-Trigerred Architecture* (TTA) todas informações sobre o comportamento do sistema, tal como, nós transmissores e receptores em um determinado instante no tempo são definidas a priori. Em fase de projeto. Dois tipos de frames são definidos no protocolo. O primeiro, denominado *I-Frame* (*Initialization frames*), é usado para inicializar o sistema. Ele contém o estado interno do controlador TTP/C em seu campo de dados. Este permite integrar nós para participar do protocolo quando recebem um *I-Frame*. *I-Frame* são enviados pelo sistema de comunicação TTP/C durante a fase de inicialização do protocolo (*cold start*), e em intervalos predefinidos durante a operação normal do protocolo para facilitar a reintegração de nós em estado de falha. Por fim, o *N-Frame* (*Normal frame*) são usados durante a operação normal e contém dados do software de aplicação.

O byte de cabeçalho de um *N-Frame* consiste em três campos, o primeiro bit identifica o tipo da mensagem, os três bits seguintes são usados para requisição de troca de modo de operação do sistema como todo, e os demais quatro bits são usados como informação de reconhecimento sobre recebimento da mensagem.

O controle de acesso ao barramento é realizado por um esquema estático TDMA. Cada nó tem permissão de transmissão somente durante uma janela de tempo determinada, denominada de janela TDMA. Com relação a duração das janelas TDMA e a sequência de envio dos nós, todos janelas TDMA são iguais. Entretanto, o comprimento e conteúdo das mensagens se diferem a cada ciclo de acordo com o software de aplicação.

Os atributos das mensagens enviadas e recebidas pelo protocolo são descritas em uma estrutura de dados estática, denominada de *Message Descriptor List* (MEDL) que reside em memória ROM dentro do subsistema de comunicação. De acordo com esta lista o controlador TTP/C, periodicamente e autonomamente lê da CNI as mensagens a serem transmitidas e escreve as mensagens recebidas também na CNI. Um dos mais importantes dados presentes na MEDL é, entretanto, o endereço de cada mensagem dentro da CNI e o comprimento da mesma.

2.2.6 FlexRay

O FlexRay é uma plataforma de comunicação completa, com alta taxa de transferência, determinístico e ainda suporta técnicas de tolerância a falhas. É aplicável em sistemas de segurança critica em automóveis. Foi desenvolvido em um consorcio de grandes empresas da área automotiva, *Tear* 1 e 2, e empresas da área de semicondutores. As principais empresas participantes do *core partners* são: BMW, DaimlerChrysler, General Motors, Ford, Volkswagen, Bosch, Motorola e Philips.

O FlexRay não substitui os demais protocolos de aplicação automotiva, ao contrário, ele opera coexistindo com protocolos existentes desempenhando sua função especifica, como o CAN, LIN e MOST. A Figura 11 apresenta um diagrama de blocos de arquitetura eletroeletrônica com FlexRay e outros protocolos.

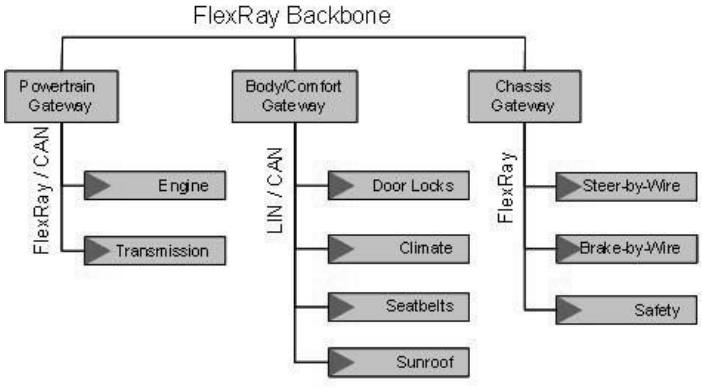

Figura 11: Exemplo de uma arquitetura com FlexRay

A troca de dados entre numerosos dispositivos de controle, sensores e atuadores nas aplicações SDTR em automóveis é, atualmente, realizada através do protocolo CAN. Entretanto, a introdução dos novos conceitos, como os sistemas *x-by-wire*, resultaram no aumento dos requisitos, especialmente com relação a tolerância a falhas, erros e determinismo temporal nas transmissões de mensagens. O FlexRay comporta estes requisitos através de transmissão de mensagens em janelas fixas e pelas técnicas de tolerância a

falhas e redundância nas transmissões em dois canais físicos.

O FlexRay trabalha com o principio TDMA, onde as mensagens têm suas janelas de transmissão fixas nas quais cada nó tem exclusivo acesso ao barramento neste instante. As janelas são repetidas por ciclos. O tempo no qual cada mensagem é transmitida é previsível, consequentemente é um meio de comunicação determinístico.

Entretanto, as janelas de transmissões fixas para cada mensagem têm desvantagem, que é a não exploração eficiente do recurso físico, como citado das seções anteriores. Por esta razão, o FlexRay subdivide o ciclo em dois segmentos, um estático e outro dinâmico. As janelas fixas são situadas no segmento estático no inicio de cada ciclo de transmissão. No segmento dinâmico as janelas são designadas dinamicamente para cada nó. Exclusivo acesso ao barramento é somente permitido em um curto tempo (denominado *mini-slots*). O tempo das janelas, neste segmento, é somente estendido para o tempo requerido em projeto se, somente se, ocorrer transmissão dentro do *mini-slot*. Desta forma, FlexRay proporciona uma melhor utilização do recurso, sendo usado somente com necessidades de transmissão. A Figura 12 apresenta o ciclo de comunicação com os dois segmento, estático e dinâmico.

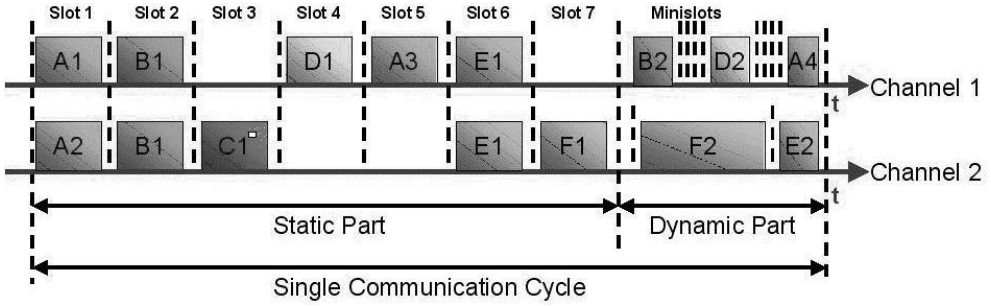

Figura 12: Segmentos estático e dinâmico do protocolo FlexRay (CONSORTIUM, Copyright 2004)

A plataforma de comunicação do FlexRay possui dois canais físicos separados, com taxa de transferência de até 10 Mbit/s cada. Os dois canais são usados redundantemente, mas pode também transmitir mensagens diferentes, em tal caso a vazão de informação (*throughput*) é dobrada (Figura 12). O FlexRay pode também operar com baixa taxa de transmissão.

Para o suporte a funções síncronas e otimização de largura de banda por meio de pequenas distâncias entre mensagens, o protocolo implementa uma base de dados comum (*global time*). A sincronização de relógio é realizada com a transmissão de uma mensagem no segmento estático do ciclo, e, com base instante de recebimento desta mensagem, os relógios locais de cada nó da rede FlexRay são atualizados.

Uma ECU FlexRay consiste em um processador, o controlador de comunicação (CC) e o guardião de barramento (*bus guardian* - BG). O processador processa dados do software de aplicação e fornece-os para o controlador de comunicação que, por sua vez, se encarrega de transmiti-los. Os guardiões de barramento monitoram o acesso ao barramento de comunicação. O processador informa ao BG em qual janela de transmissão o CC do respectivo nó está alocado. O BG, então, permite o CC transmitir a mensagem somente nesta janela. O recebimento de dados é sempre permitido.

3 PROTOCOLO FTT-CAN

Flexible Time-Triggered on Controller Area Network (FTT-CAN) foi proposto com o objetivo de cobrir o requisito de flexibilidade em sistemas de tempo-real. Tal requisito não é presente em protocolos essencialmente TT. Basicamente, o protocolo faz uso do conceito de ciclo elementar com duas fases de transmissão a fim de combinar ambos os paradigmas TT e ET com um isolamento temporal entre eles. Além disso, o tráfego TT é escalonado *on-line* por um nó particular denominado de nó mestre, que provê um controle de admissão de mensagem no segmento TT sem prejudicar os requisitos temporais das mensagens já presentes no sistema (ALMEIDA; PEDREIRAS; FONSECA, 2002). O controle de admissão é realizado por um escalonador central, executado pelo nó mestre. Este pode receber de forma dinâmica, ou seja, em tempo de execução, novos pedidos de transmissão de mensagens TT não antes previstas. Estas requisições são processadas e avaliadas por um algoritmo que concede a factibilidade da nova inserção. Com controle de admissão online, o protocolo suporta o tráfego de TT em um modo flexível, sob a garantia dos requisitos temporais (baseado no modelo de escalonamento dinâmico).

O FTT-CAN aproveita-se do nativo controle de acesso ao meio do protocolo CAN para eliminar a necessidade de informações de controle no protocolo a fim de controlar a comunicação TT. Desta forma, reduzindo o overhead do protocolo. O nó mestre ativa as transmissões nos nós escravos seguindo um modelo denominado mestre-escravo flexível (traduzido do termo em inglês *relaxed master-slave method*), que requisita de forma simultânea as mensagens a serem transmitidas em um dado segmento TT de um ciclo elementar. Uma mensagem específica - denominada de *Trigger Message* (TM) - é transmitida pelo nó mestre a fim de ativar o início de um ciclo elementar ou *Elementary Cycle* (EC) dentro de cada nó escravo que por sua vez transmitirá mensagens nos segmentos TT e ET. A mensagem TM transporta informações que indicam quais mensagens serão transmitidas no segmento TT. A Figura 13 apresenta um ciclo EC com seus respectivos segmentos, e exemplifica a codificação de uma mensagem TM.

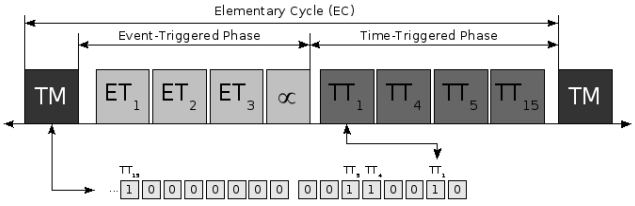

Figura 13: Ciclo elementar (EC) do protocolo FTT-CAN

Em cada EC o protocolo define duas fases sucessivas, assíncronas e síncronas, que corresponde a dois segmentos separados. O primeiro usa para comunicação ET, chamado

assíncrono porque os pedidos de transmissão podem ser emitidos a qualquer momento, gerado, por exemplo, devida uma troca de estado de algum sensor monitorado. O posterior é usado para comunicação TT, chamado síncrono porque as transmissões são realizada de forma síncrona em relação a mensagem TM enviada pelo nó mestre em cada EC. O segmento síncrono ou TT do EC tem uma duração $lsw(n)$ e é ajustada de acordo com o tráfego escalonado para o segmento no respectivo EC. O segmento assíncrono ou ET tem uma duração $law(n)$ igual ao restante de tempo entre a mensagem TM e o segmento TT. Ou seja, o segmento TT tem seu $lsw(n)$ variável, de acordo com o número de mensagens no respectivo EC. O protocolo permite estabelecer uma duração de máximo para o segmento TT e correspondentemente uma largura de banda máxima para este tipo de tráfego. O restante é destinado para o segmento ET.

Não existem requisições explicitas por mensagem do mestre para os nós escravos, desta forma cada nó escravo trabalham de forma independente seguindo somente a mensagem TM no inicio de cada ciclo EC. As eventuais colisões entre mensagens dos nós escravos em ambos os segmentos são resolvidas pelo nativo mecanismo de arbitramento do protocolo CAN. Vale ressaltar, que o segmento TT possui somente os instantes de inicio e fim sem as divisões de tempo ou janelas de transmissão como existe outros protocolos. De forma, todas as mensagens TT são transmitidas no mesmo instante tempo no inicio do segmento e ordenadas através do mecanismo de arbitramento. No segmento ET os nós com transmissões pendentes podem iniciar as transmissões imediatamente no inicio do segmento.

Para garantir as restrições temporais das mensagens do segmento TT o protocolo FTT-CAN o protege de interferências oriundas de requisições assíncronas dentro do segmento TT. Para isto, um isolamento temporal entre ambos os segmentos é forçado, prevenindo a tentativa de transmissões que não possam ser completadas dentro do segmento ET. Este isolamento é alcançado removendo do *buffer* de transmissão do controlador de rede qualquer pedido pendente que não possa ser realizado até conclusão daquele segmento, mantendo-os em filas de transmissão para o próximo segmento ET. Deste modo, uma pequena quantia de tempo de inatividade pode surgir no fim do segmento ET. Por outro lado, no fim do segmento TT, outra pequena quantia de tempo de inatividade surge devido as variações geradas pelo mecanismo de *bit stuffing* usado na codificação física do protocolo CAN. Relacionado ao efeito do mecanismo de *bit stuffing*, os autores (ALMEIDA; PEDREIRAS; FONSECA, 2002) consideram o pior caso de ocorrência. Isto negligencia uma fonte de *jitter* no segmento TT. A seção 4 discute este problema que afeta o comportamento temporal deste segmento.

O protocolo FTT-CAN possui dois serviços de comunicação, uma para cada segmento. *Synchronous Messaging System* (SMS) e *Asynchronous Messaging System* (AMS). O serviço de SMS segue o modelo produtor-consumidor enquanto o AMS oferece somente os serviços básicos de transmissão e recepção usados quando a camada de aplicações possui mensagens aperiódicas para transmissão. Os nós escravos com mensagens aperiódicas pendentes podem transmitir somente durante o segmento ET que é o tempo restante não usado pelo segmento TT no ciclo EC. As transmissões de mensagens periódicas são realizadas de maneira autônoma, no ponto de vista do software de aplicação. Isto é, o protocolo é responsável pela transmissão de todas as mensagens dentro do segmento TT, de forma que as tarefas do software de aplicação não necessitam invocar os serviços de envio e recepção de mensagens. No segmento ET, porém, mensagens são transmitidas em resposta para pedidos de explícito da camada de aplicações.

3.1 Synchronous Messaging System

A duração do ciclo EC é uma unidade de tempo usado como atributo de controle do tráfego de mensagens síncronas. Sua duração segue o menor período do conjunto de mensagens. Ou seja, o ciclo EC terá sua duração determinada pelo menor período das mensagens TT. Depois da transmissão da mensagem TM todos os nós que produziram mensagens TT devem decodificar essa mensagem a fim de checar se os mesmos serão produtores ou consumidores de alguma mensagem codificada no campo de dados da mensagem TM. Esta verificação é realizada por cada nó da rede varrendo a tabela local onde contém a identificação das mensagens a serem produzidas e ou consumidas pelo nó.

O controle de acesso de meio nativo do protocolo de base CAN permite realizar um escalonamento centralizado com baixo consumo de recursos computacionais, ou seja, uma mensagem TM por EC. Não existe a necessidade de indicar o tempo de transmissão de mensagens de TT no seu respectivo segmento no ciclo EC, isto é resolvido pelo mecanismo de arbitragem do protocolo CAN. A Tabela 1, como apresentada em (ALMEIDA; PEDREIRAS; FONSECA, 2002), indica a largura de banda usada por TM em quatro situações. Para cada taxa de transmissão, a consumo de recurso computacional ou *overhead* pode ser reduzido com o aumento da duração do clico EC (E), ou reduzindo o comprimento de dados da mensagem TM (LTM).

Tabela 1: Overhead da mensagem TM

Tx Rate (Mbits/s)	TM data length (bytes)	$LTM(\mu s)$	$E(ms)$	Overhead (%)
0,125	4	736	10	7,4
0,125	8	1040	10	10
1	4	92	5	1,8
1	8	130	5	2,6

3.1.1 Requisitos e escalonamento no SMS

Todos os atributos das mensagens TT são armazenados em uma tabela no nó mestre denominada de *Synchronous Requirements Table* (STR).

$$SRT \equiv \{SM_i(DLC_i, C_i, Ph_i, P_i, D_i, Pr_i), i = 1...N_s\}$$

Onde DLC é o comprimento de dados em bytes, que segue o comprimento de dados de mensagem CAN, C é o tempo de transmissão máximo de uma mensagem considerando, inclusive, a ocorrência no pior caso de *bit stuffing*, Ph indica o deslocamento de fase da mensagem em unidades de tempo baseada no período do ciclo EC, P para período, D para o *deadline*, e Pr para prioridade. Os atributos P e D são expressos como inteiro múltiplos da duração do ciclo EC. Ns é o número de mensagens TT presente na SRT. Um escalonador *on-line*, residente no nó mestre, constrói o plano de escalonamento TT baseado-se nos atributos contidos na SRT. Este gera um plano para cada ciclo EC codificando-o nos bits do campo de dados da mensagem TM, que é transmitido em *broadcast* na rede no inicio de cada EC. Esta característica do FTT-CAN permite um trafego de mensagens TT com atrasos conhecidos e de forma flexível ou adaptativo.

Duas abordagem diferentes foram proposta para a implementação do escalonador *on-line* no nó mestre. A primeira é o *planning-scheduler* apresentado em (ALMEIDA; PASADAS; FONSECA, 2001), trata-se de uma implementação baseada em software que

permite reduzir o consumo de recurso de processamento do escalonador *on-line*. Esta abordagem constrói uma tabela de plano estático para um dado período de tempo denominado *plan* e reconstrói essa tabela *on-line* no fim de cada *plan*. A duração do *plan* não é relacionada aos períodos de mensagens e deste modo os requisitos de memória para armazenar uma tabela *plan* são bem limitados e conhecidos a priori. Por outro lado, o *plan*, uma vez construído não pode ser alterado. Assim, o *planning-scheduler* estabelece um compromisso entre o consumo de recursos computacionais e as reatividades de mudanças *on-line*, e é adequado para sistemas com baixo poder computacional. Por exemplo, nós baseados em microcontroladores de 8bits. Uma desvantagem deste modelo de escalonamento *on-line* e seu baixo tempo de resposta às requisições de mudanças dos atributos das mensagens TT. A outra abordagem desenvolvida (MARTINS; ALMEIDA; FONSECA, 2005) faz uso de co-processador baseado em FPGA. Esta abordagem provê, em um alto custo de hardware, uma capacidade computacional extra requerida para executar a política de escalonamento *on-line* como também uma verificação de escalonabilidade. Por exemplo, o escalonador verifica a SRT e cria um novo plano a cada EC. Além disso, é capaz de executar testes de escalonabilidade neste intervalo de tempo. O resultado desta solução é um alto grau de flexibilidade e tempo de resposta as requisições de mudança dos atributos das mensagens TT.

Como o plano de escalonamento é feito baseada nos atributos da SRT independentemente do identificador das mensagens, qualquer política de escalonamento pode ser facilmente implementada no nó mestre. Por exemplo: *Rate-Monotonic* (RM), *Deadline-Monotonic* (DM), *Earliest-Deadline First* (EDF) e *Least-Laxity First* (LLF). Os autores apresentam em (ALMEIDA; PEDREIRAS; FONSECA, 2002) uma análise do uso do FTT-CAN com RM e EDF, usando 80% da largura de banda do barramento alocado para o SMS ($LSW = 0,8xE$). A Figure 14 apresenta um resumo deste trabalho, indicando a porcentagem de um conjunto de mensagem fictício escalonado em ambas as políticas em função do fator de utilização do barramento. Com a política de escalonamento EDF, praticamente toda a largura de banda alocada para o SMS poderia ser usado com garantias de escalonabilidade. O fato dessa política não alcançar 100% de utilização de largura de banda alocada é explicado pela interferência da impossibilidade de preempção na transmissão de mensagem.

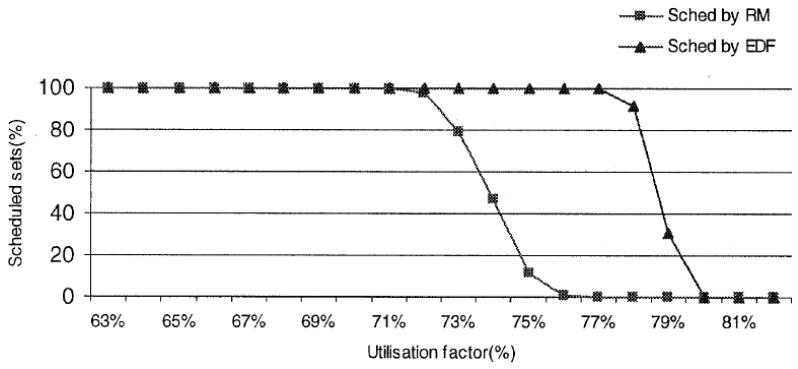

Figura 14: Escalonabilidade versus utilização do barramento nas políticas RM e EDF (ALMEIDA; FONSECA, 2001)

O modelo usado para escalonar o tráfego de mensagem TT é similar ao apresentado em (ALMEIDA; FONSECA, 2001), chamada *blocking-free nonpreemptive scheduling*. A única diferença é que, em (ALMEIDA; FONSECA, 2001), todo o ciclo é disponível para

execução de tarefas enquanto que no FTT-CAN o trafego de mensagens TT é restrito ao seu respectivo segmento dentro do ciclo EC com comprimento máximo LSW. Assim os períodos de mensagem TT e os *deadlines* são inteiros múltiplos da duração do ciclo EC, os tempos de transmissão são sempre menores que o ciclo EC e as ativações de mensagens são sempre síncronos com o inicio do segmento TT.

Para uma análise de escalonabilidade, é necessário modelar o efeito da mensagem TM e a limitação no comprimento do segmento TT, que é restrito a uma fração do ciclo EC. Primeiramente todos os tempos de transmissão devem ser inchados por um fator igual a $\frac{E}{LSW}$. A Figura 15 ilustra este processo. Isto é equivalente a expandir o segmento TT no comprimento do ciclo EC, fazendo uma ilusão do segmento como um tempo total disponível.

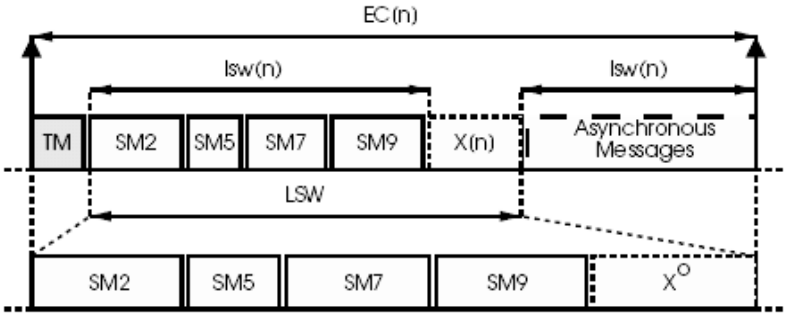

Figura 15: Expansão da fase TT (ALMEIDA; PEDREIRAS; FONSECA, 2002)

Aplicando esta condição no conjunto original de mensagem da SRT resulta em um novo conjunto virtual, chamado SRT^o, onde os tempos de transmissão são atualizados.

$$SRT^o \equiv SM_i^o(DLC_i, C_i^o, Ph_i, P_i, D_i, Pr_i), C_i^o = \left(\frac{E}{LSW}\right) * C_i, i = 1...N_s$$

Deste modo, o resultado em (ALMEIDA; FONSECA, 2001) são agora aplicáveis sobre a SRT^o, particularmente o teorema declara que qualquer análise baseada em prioridade fixa pode ser usado neste modelo se o tempo de transmissão C_i^o é substituído pelo C_i'.

$$C_i' = \frac{C_i^o * E}{E - X^o}$$

Onde E é a duração do ciclo EC e X^o é o máximo tempo de inatividade (*idel time*) considerado. Agora, expandindo a equação com SRT^o e notando que o $X^o = \frac{E}{LSW} * X$, produz uma equação final que deve ser realizada sobre os tempos de transmissão de mensagens originais, aqueles da SRT, de forma que qualquer análise existente de prioridade fixa preemptivo pode ser usada.

$$C_i' = C_i * \left(\frac{E}{LSW} - X\right)$$

O termo X introduz um pessimismo quando usado como limite superior, que pode ser, por exemplo, o tempo de inatividade causado por uma mensagem com tempo de transmissão longo no conjunto de mensagens TT. A utilização do limite superior da política *Rate Monotonic* (LIU; LAYLAND, 1973) pode ser usado com uma adaptação como parte de um controle de admissão *on-line*. Isto é expresso como:

$$\sum_{i=1}^{N_s} \left(\frac{C_i}{P_i}\right) < N_s \left(2^{\frac{1}{N_s}} - 1\right) * \left(\frac{LSW - X}{E}\right)$$

Se esta condição é verdadeira, então SRT é escalonável com a política RM.

Em modo semelhante, o limite de utilização com EDF (LIU; LAYLAND, 1973) pode ser usado. Neste caso, o tempo máximo de inatividade (X) inserido somado a quantia restante de tempo do ciclo EC fora do segmento TT ($E - LSW$) pode ser considerado como o pior caso de tempo de transmissão de uma mensagem virtual v, com pior caso de tempo de transmissão $C_v = E - LSW + X$, que é adicionado ao conjunto original de mensagens transmitidas em cada ciclo EC ($P_v = 1EC$). A mensagem virtual v tem alta prioridade, desde que $P_v = D_v = 1EC$, e preenche a parte do ciclo EC que não pode ser usado pelo tráfego de mensagens TT. A Figura 16 apresenta o período Pv da mensagem virtual. Agora, assuma que o conjunto estendido resultante, o SRT original e mensagem virtual v, podem ser escalonados considerando preempções. Devido à ausência de pontos de preempção, motivados pela aparente ativação síncrona, e devido à ausência de efeito de bloqueio pela inserção do tempo de inatividade, o limite de utilização pode ser usado:

$$U_v = \left(\frac{E - LSW + X}{E}\right) + \sum_{i=1}^{N_s} \left(\frac{C_i}{P_i}\right) \leq 1$$

Esta condição acima apresenta limite de utilização considerando a mensagem virtual v e a STR.

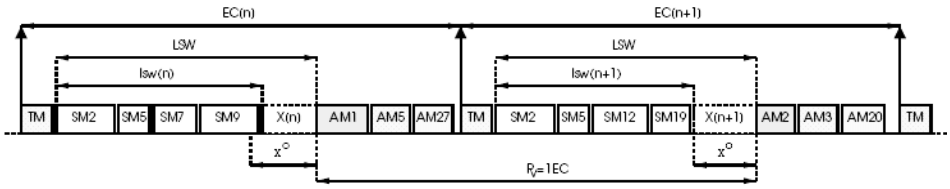

Figura 16: Efeito da inserção do tempo inativo, fase ET e TM (ALMEIDA; PEDREIRAS; FONSECA, 2002)

Devido à carga extra imposta pela mensagem virtual v, todas outras mensagens terminarão a transmissão ou no mesmo EC ou mais adiante nesta estratégia de escalonamento do que no conjunto original com o tráfego limitado no segmento TT e com o tempo de inatividade inserido. Se o conjunto estendido é escalonável a SRT também será. A condição final de escalonabilidade é:

$$\sum_{i=1}^{N_s} \left(\frac{C_i}{P_i}\right) \leq \left(\frac{LSW - X}{E}\right)$$

Se esta condição é verdadeira, então o SRT é escalonável com a política EDF. Novamente, existe um pessimismo relacionado ao tempo de inatividade máximo inserido, deste modo levando para uma análise que é suficiente.

3.2 Asynchronous Messaging System

No segmento de trafego assíncrono é realizada a comunicação ET, o qual é tratado pelo serviço AMS. Este subsistema do protocolo FTT-CAN trabalha semelhantemente a sistemas inteiramente baseados no protocolo CAN usando seu mecanismo de arbitramento distribuída baseada em prioridade. Porém, no protocolo FTT-CAN, o AMS contém um nível de controle de acesso que permite restringir este tipo de tráfego assíncrono somente para o seu respectivo segmento no ciclo EC. Isto é importante para prevenir mensagens ET interfiram no segmento TT ou durante o instante de tempo de transmissão da mensagem TM. Assim forçando o isolamento temporal entre ambos os segmentos. O controle de acesso estabelece o início e fim de cada segmento ET é baseado somente no tempo relativo à recepção da mensagem TM, sem qualquer outro controle baseado em trocas de mensagem.

Os nós escravos com mensagens ET para transmissão tentam imediatamente transmitir durante seu respectivo segmento. Por outro lado, se alguma mensagem fica pendente devido ao fim do segmento, esta é mantida em uma fila até o próximo segmento ET, então participando novamente do processo de arbitramento. Esta coexistência de tráfego ET e TT que motiva a divisão em segmentos específicos para cada um introduz uma redução do tempo de resposta dos pedidos de transmissões oriundas a ocorrências de eventos assíncronos. No pior caso, é necessário considerar as condições de bloqueio causadas pelos períodos de exclusão de barramento. Isto é, os períodos de tempo fora do segmento ET.

De acordo com a especificação do protocolo FTT-CAN, toda atividade de comunicação no sistema AMS segue o paradigma de controle externo, onde a transmissão de mensagens acontece através de pedidos explícitos do software de aplicação através das chamadas de função AMS_{send} e $AMS_{receive}$. Uma fila é ordenada primeiramente por prioridade, de acordo com o identificador da mensagem, e, segundo por momento de pedido (FCFS - *first come, first serve*). O comprimento da fila dentro de cada nó é configurado em tempo de projeto de acordo com o número de mensagens ET que o nó pode transmitir considerando o número de mensagens do mesmo fluxo que este pode enfileirar ao mesmo tempo.

A entrega de mensagens para o software aplicação é realizada por meio de uma chamada de serviço, $AMS_{receive}$, que permite a espera por uma mensagem especifica. No nó receptor, o serviço AMS enfileira as mensagens que chegam da rede até que elas sejam capturadas pelo software de aplicação. O comprimento da fila também configurado em tempo de projeto, semelhantemente a fila no lado do transmissor de mensagem. Os aspectos importantes são os números de mensagens ET que um nó pode receber como também o número de mensagens que podem chegar ao mesmo segmento.

O serviço AMS segue o paradigma dinâmico melhor esforço. Para um dado conjunto de requisitos de comunicação, pode ser mostrado que o pior caso de tempo de resposta para pedidos de transmissão ET é limitado, deste modo suportando restrições temporais nas transmissões ET (por exemplo, alarmes). A política de escalonamento usada é herdada do protocolo CAN original, baseado em prioridade fixa. O AMS usa a inserção de tempo de inatividade para forçar o isolamento temporal entre os dois tipos de tráfego. A largura de banda usada para o segmento ET é inversamente proporcional o segmento TT. Quando há um segmento TT mais longo, menor será o ET.

Um conjunto de requisitos de comunicação de tempo real ET é inserido em uma tabela chamada ART (*Asynchronous Requirements Table*):

$$ART \equiv \{AM_i(DLC_i, C_i, mit_i, D_i, Pr_i), i = 1...N_a\}$$

Toda mensagem nesta tabela é caracterizada como esporádica, com um intervalo de tempo mínimo (mit_i) que deve decorrer entre mensagens sucessivas do mesmo fluxo. Os parâmetros DLC, C, e D são equivalente aos da mensagem síncrona a não ser que este *deadline* não seja um inteiro múltiplo de E. Pr é a prioridade de mensagem, que é diretamente expresso com o identificador de mensagem CAN. N_a é o conjunto de mensagem ET. A análise de mensagens assíncronas pode ser feita considerando os requisitos temporais, onde pode ser definido dois conjuntos ART, um N_a^{RT} e outro N_a^{nRT}. Ou ainda, apenas considerar mensagens não de tempo real como de menor prioridade ao conjunto de mensagens original.

3.3 Escalonamento de tarefas em um sistema FTT-CAN

Da mesma forma que o escalonamento de mensagens TT é realizado no protocolo FTT-CAN, as tarefas também podem ser despachadas (CALHA; FONSECA, 2002). Deste modo o nó mestre assume o controle do sistema distribuído de forma global, ativando mensagens e tarefas no barramento através da mensagem TM. Esta se torna um mecanismo confiável, provendo um sincronismo global entre nós presentes na rede. Este mecanismo garante a viabilidade do propósito de escalonamento de tarefas no sistema sem significantes consumo de recursos computacionais.

Em SDTR tarefas podem produzir ou consumir uma mensagem, ou em alguns casos ambos. Uma tarefa que gera algum dado é chamada uma tarefa produtora e por outro lado uma tarefa que usa dados para qualquer propósito é chamada de tarefa de consumidor, que constitui a estratégia produtor-consumidor. As demais tarefas do sistema que não interagem com outras, são, deste modo, chamadas de tarefas independentes ou *stand-alone*. As interações entre tarefas podem ser representadas como gráficos de precedência, que mostram as relações de dependências entre tarefas e mensagens, e também mostram o fluxo de dados entre esses. O nó de mestre pode controlar a execução de tarefas e mensagens associadas com o fluxo de dados do produtor até as tarefas consumidoras. Deste modo o proposta FTT, quer seja sobre CAN ou Ethernet, pode garantir o comportamento de tempo real da execução de tarefas e transmissão de mensagens de forma integrada. Em (CALHA; FONSECA, 2002) e (CALHA; SILVA; FONSECA, 2006) os autores consideram tarefas TT escalonadas juntas a mensagens TT.

Para este propósito o campo de dados da mensagem TM deve acomodar duas áreas de flags de ativação, uma para tarefas e outras para mensagens. Como os flags de mensagens TT, cada bit da área separada para codificação de tarefas TT indica se uma tarefa deve ser despachada no ciclo EC corrente ou não. Para exemplificar, considerando o conjunto de tarefa apresentado (CALHA; FONSECA, 2002) e (CALHA; SILVA; FONSECA, 2006), a Figura 17 apresenta um simples gráfico de precedência com 6 tarefas que executam em 4 nós e 3 mensagens.

Neste exemplo é importante definir o deslocamento de fase relativa, Ph, de cada tarefa e mensagem. Na primeira série da Figura 17, a mensagem M1 é produzida pela tarefa T1 e consumida pela tarefa T2. A mensagem M2 é produzida pela tarefa T2 e consumido por ambas as tarefa T3 e T4. No segundo fluxo, a mensagem M3 é produzida pela tarefa T5 e consumida pela tarefa T6. De acordo com o conjunto de tarefas e mensagens da Figura 17, a mensagem TM poderia ter um campo de dados com 2 bytes, sendo um byte para

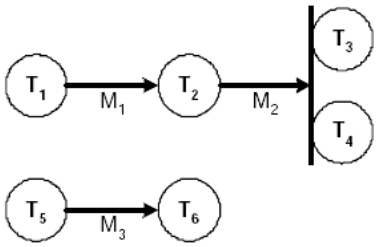

Figura 17: Exemplo de gráfico de precedência (CALHA; FONSECA, 2002)

tarefas e uma outro para mensagens. Esta definição é feita durante a fase de projeto. Se a soma de tarefas e mensagens excede 64 então uma mensagem TM extra é necessário, devido o limite de 64 bits de dados em uma mensagem CAN. A Figura 18 mostra as mensagens TM para o conjunto de tarefas e mensagens da Figura 17.

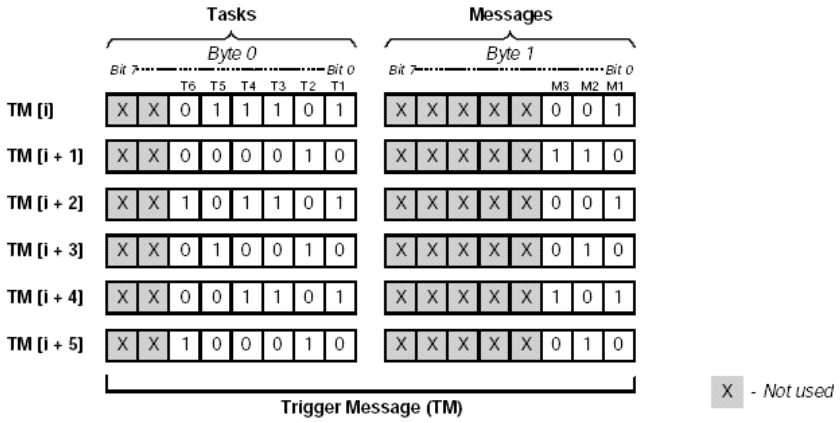

Figura 18: Mensagem TM com tarefas e mensagens codificadas (CALHA; FONSECA, 2002)

Uma restrição imposta pela mensagem TM é que todo período, P, e fase, Ph, para ambas as tarefas e mensagens têm que ser arredondadas para um valor múltiplo do ciclo EC. A duração de ciclo é a unidade básica de tempo para o sistema.

Agora, o nó mestre necessita negociar mensagens e tarefas, então o STR deve ser atualizado com os atributos das tarefas TT. Uma tarefa TT produtora ou consumidora tem o mesmo período de sua respectiva mensagem. Um conjunto destas tarefas tem alguns atributos.

$$SRT \equiv \{ST_i(N_i, MP_i, MC_i, C_i, Ph_i, P_i, D_i, Pr_i), i = 1...N_{st}\}$$

Onde ST_i representa umas tarefas TT, C é o tempo de execução no pior caso, P o período, D o *deadline* medido relativo ao instante de liberação, Pr a prioridade, N o nó onde a tarefa é executada, Ph o deslocamento de fase relativa que determina o primeiro instante de liberação após a partida do sistema. Para tarefas interativas existem ainda dois novos atributos, MP, a mensagem produzida e MC, a mensagem consumida. Os autores em (CALHA; SILVA; FONSECA, 2006) consideram o segmento de execução como o intervalo entre a liberação e o *deadline* da tarefa. No conjunto de atributos da mensagem TT dois novos atributos são necessárias.

$$SRT \equiv \{SM_i(PT, CTL_{i,j}, DLC_i, C_i, Ph_i, P_i, D_i, Pr_i), i = 1...N_{sm}\}$$

O primeiro é o PT, que representa a tarefa produtora e outro é CTL, que consiste em uma lista de tarefas consumidoras. Esta lista é necessária porque mais de uma tarefa pode consumir a saída de uma tarefa produtora.

Também, os autores em (CALHA; SILVA; FONSECA, 2006) consideram o segmento de transmissão como o intervalo entre a liberação e o *deadline* da mensagem.

Atualmente, em literaturas sobre sistemas FTT existem duas abordagens para transmissão de mensagens e ativação de tarefas. Uma abordagem denominada de *Net-Centric*, onde mensagens impõem restrições sobre o conjunto de tarefas, e outra denominada *Node-Centric* onde tarefas impõem restrições sobre o conjunto de mensagens. O principal fator é o uso dos recursos de sistema, a rede para transmissão de mensagem e os nós para execução de tarefa. De acordo com estes fatores, quatro combinações podem então ocorrer:

- Baixa carga na rede e baixa carga computacional no nó, onde qualquer abordagem pode ser usada;
- Alta carga na rede e baixa carga computacional no nó, correspondem a abordagem Net-Centric;
- Baixa carga na rede e alta carga computacional no nó, correspondem a abordagem Node-Centric;
- Alta carga na rede e alta carga computacional no nó, onde ambas as abordagens deveriam ser consideradas a fim de selecionar uma.

A respeito de escalonamento holístico (*holistic scheduling*), o ponto de partida é verificar o fluxo de dados do sistema. Após a construção do gráfico de relação de precedência a fim de definir Ph para mensagens sem considerar tarefas, o macro ciclo pode ser calculado. O macro ciclo é um intervalo que compreende um ou mais ciclos EC, com um conjunto de tarefas e mensagens, que será indefinidamente repetido, até o fim de uma tarefa ou a ocorrência de um erro (CALHA; FONSECA, 2002). Para o propósito de escalonamento, qualquer algoritmo pode ser usado, no entanto esse assunto não será discutido nesta dissertação. Existem duas abordagens para escalonamento de tarefas e mensagens, que são: escalonamento independente e dependente. O escalonamento pode ser simultaneamente realizado para tarefas e mensagens considerando escalonamento independente ou o escalonamento de mensagens é realizando antes do escalonamento de tarefas na abordagem dependente. A próxima seção apresenta detalhes da abordagem *Net-Centric* com escalonamento independente.

3.3.1 Abordagem Net-Centric independente

O autor em (CALHA; FONSECA, 2002) apresenta que tarefas são iniciadas considerando que as mensagens podem ser transmitidas a qualquer momento, no segmento de transmissão, até o *deadline*. Também, tarefas podem ser executadas a qualquer momento em sua janela de execução. Considerando a abordagem *Net-Centric* algumas restrições se aplicam na dependência das mensagens produzidas e/ou consumidas. Isto é apresentado nas seções seguintes.

3.3.1.1 Tarefa produtora

As definições principais para tarefa produtora são:

$$P_{task} = P_{msgP}$$

$$Ph_{task} < Ph_{msgP}$$

$$Ph_{msgP} = Ph_{task} + D_{task}$$

A última equação garante que, quando o *deadline* da tarefa é alcançado, a mensagem para transmissão já tenha sido produzida.

$$D_{task} = P_{task} - D_{msgP} + C_{task}$$

$$Ph_{task} = Ph_{task} + D_{task}$$

Pelas últimas equações, pode-se entender que quando o D_{msgP} é próximo do valor de P_{msgP} , a D_{task} aborda o valor de C_{task}. Então pode-se concluir que quando a janela de transmissão da mensagem se alarga, a janela de execução da tarefa se estreita, e vice versa.

A Figura 19 apresenta uma tarefa produtora.

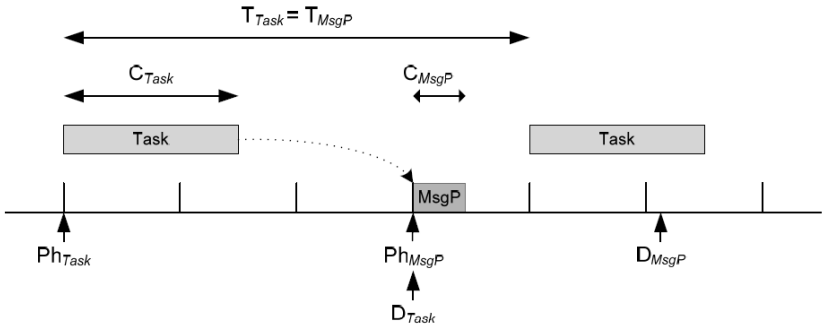

Figura 19: Tarefa produtora

3.3.1.2 Tarefa consumidora

As definições principais para tarefa consumidora são:

$$P_{task} = P_{msgC}$$

$$Ph_{msgC} < Ph_{task}$$

$$Ph_{task} = Ph_{msgC} + D_{msgC}$$

A última equação garante que quando a tarefa inicia a mensagem a ser consumida já foi transmitida.

$$Ph_{task} = Ph_{msgC} + D_{msgC}$$

$$D_{task} = P_{msgC} + C_{task} - D_{msgC}$$

Pelas últimas equações, pode-se entender que quando o D_{msgC} é próximo do valor de T_{msgC}, a D_{task} aborda o valor de C_{task}. Então podemos concluir que quando a janela de transmissão da mensagem se alarga, a janela de execução da tarefa estreite, e viceversa.

A Figura 19 apresenta uma tarefa consumidora.

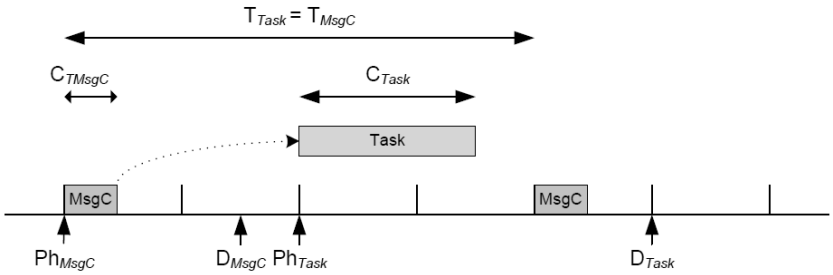

Figura 20: Tarefa comsumidora

3.3.1.3 Tarefa consumidora/produtora

As principais definições para tarefas consumidoras/produtoras são:

$$P_{task} = P_{msgC} = P_{msgP}$$

$$Ph_{msgC} < Ph_{task} < Ph_{msgP}$$

$$Ph_{task} = Ph_{msgC} + D_{msgC}$$

$$D_{task} = Min(D_{msgP}, D_{msgC})$$

$$Ph_{msgP} = Ph_{task} + D_{task}$$

O instante de liberação da tarefa (Ph_{task} no caso da primeira ativação), deve acontecer no EC depois do *deadline* da mensagem sendo consumida, D_{msgC}. Isto garante que a mensagem foi transmitida e os dados para serem consumidos já foram entregues para o software de aplicação quando a mensagem TM chega para disparar as tarefas. O *deadline* da tarefa, D_{task}, que é medido relativo ao instante de liberação da tarefa, deve acontecer no ciclo EC antes do instante de liberação da mensagem sendo produzida (Ph_{msgP} no caso do primeiro ativação) e antes também do instante em que seria possível a próxima mensagem a ser consumida pela tarefa atual. Dependendo dos *deadlines* das mensagens produzida e consumida, a mensagem com a maior janela de transmissão prevalecerá quando definindo a janela de execução de a tarefa.

A Figura 19 shows um consumer/producer tarefa.

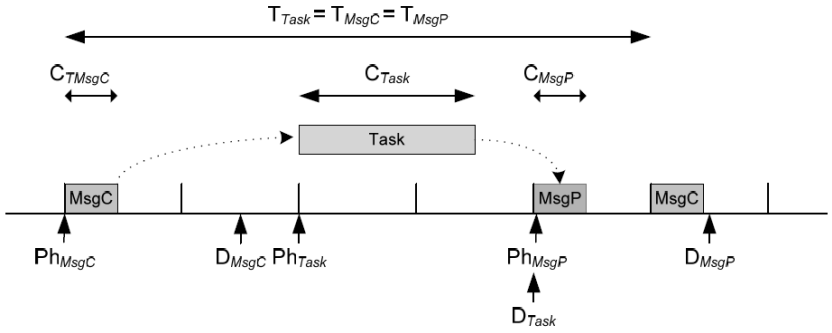

Figura 21: Tarafa comsumidora/produtora

4 PROPOSTAS PARA O PROTOCOLO FTT-CAN

Alguns problemas que têm impacto sobre o desempenho de sistema de controle via rede baseado em protocolo FTT-CAN foram identificados durante esse trabalho. Estas desvantagens são apresentados nas seções a seguir. O primeiro foco é na transmissão síncrona de mensagem, onde são apresentados dois aspectos negligenciados que geram *jitter* neste tráfego. O outro foco é sobre o disparo de tarefas síncronas, as quais apresentam deficiência na sua execução de tarefas síncronas, sendo que este aspecto ainda não havia sido abordado na literatura.

Antes da apresentação dos pontos citados acima apresenta-se uma extensão do estudo comparativo realizado em (LONN; AXELSSON, 1999). É uma comparação entre escalonamento por prioridade fixa (*Fixed Priority*) e estático cíclico (*Static Cyclic*) de tarefas e mensagens relativas as características temporais em uma aplicação de sistema controle via rede. Diferente do estudo de (LONN; AXELSSON, 1999), na comparação que será apresentada, analisar-se-á o impacto do método *bit stuffing* do protocolo CAN na variação do tempo de transmissão de mensagens. Esta análise é apresentada juntamente a abordagem de *offset* (TINDELL; CLARK, 1994) e considerando suporte a tempo global de sistema.

4.1 Escalonamento de Tarefas e Mensagens

A maioria dos sistemas embarcados têm que satisfazer exigências rígidas de tempo. No entanto, quando tais sistemas são implementados de forma distribuída, a previsibilidade têm de ser garantida para as transmissões de mensagens e execução de tarefas. Um escalonador é um algoritmo que executa as tarefas ou transmite mensagens com consideração de algumas regras, tendo em conta o tempo e utilização de recursos. Um fator importante na fase de concepção em sistemas de tempo real distribuído (STRD) a respeito dos requisitos temporais é a escolha da política de escalonamento de tarefas e mensagens. Isto porque as políticas levam a um comportamento diferente em termos utilização de recursos, flexibilidade, e tempo de resposta à eventos externos. As abordagens de escalonamento de tarefas e mensagens podem ser classificadas como dinâmicos ou estáticos.

O escalonamento dinâmico é executado durante o tempo de execução (sendo por isto também conhecido como *on-line*) e todos os parâmetros envolvidos são variáveis de acordo com a evolução do sistema. Este tipo de escalonamento constrói em tempo de execução uma tabela de disparo com os instantes de execução das tarefas ou mensagens, com base em algumas regras da política escolhida. Este tipo de escalonamento tem um alto custo em relação a outros, porque exige que uma análise de escalonabilidade seja feita durante o tempo de execução. Todavia, por permitir uma adaptação a novos requisitos,

apresenta um alto nível de flexibilidade. Por outro lado, a estratégia de escalonamento estático é realizada em fase de Projeto (sendo por isto denominada de *off-line*) e considera todos os parâmetros em uma forma estática. A análise de escalonabilidade é realizada na fase de concepção, podendo-se definir "a priori"se o sistema atende ou não os requisitos impostos. Em uma abordagem *off-line*, podemos citar as políticas cíclico estático e a prioridade fixa, que também podem ser identificadas como abordagem TT e ET, respectivamente, tal como foi apresentado como na seção 2.1.

Considerando um SDTR, uma determinada tarefa em um determinado nó se comunica com tarefas em outros nós por meio de mensagens enviadas através de um barramento de comunicação. Essa abordagem faz com que os atributos de tarefas sejam herdadas pelas mensagens no ponto de vista do nó emissor, e por outra lado os atributos das mensagens são herdados pelas tarefas (TINDELL; CLARK, 1994). Em outras palavras, uma mensagem transmitida em um barramento de comunicação herda o *jitter*, o tempo de resposta e o período da tarefa que produziu tal mensagem. Este tipo de relacionamento entre mensagens e tarefas direciona a abordagem denominada *holistic scheduling* (POP, 2003).

A Figura 22 apresenta uma simples operação entre duas tarefas em SDTR baseado no paradigma ET, que pode ser analisada também como um laço de controle entre um nó sensor e outro atuador. A mensagem m é transmitida após a tarefa, s (tarefa de amostragem), terminar sua execução no $node_a$, após o processo de amostragem. As tarefas c, a (tarefas de controle e atuação) são ativadas logo após a recepção da mensagem m pelo $node_b$.

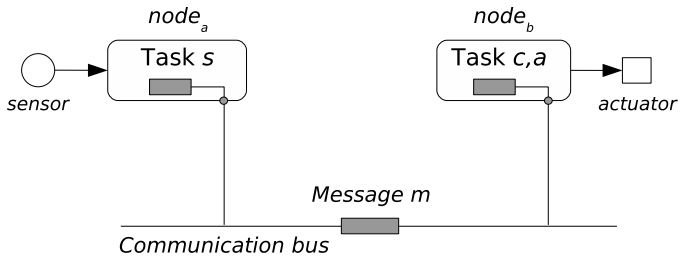

Figura 22: Herança de atributos entre tarefas e mensagens

Tipicamente, em SDTR ET uma mensagem, m_i, é enfileirada por uma tarefa t_i. Assumindo uma tarefa periódica com período T, então m_i herda o período igual ao período de t_i ($T_{t_i} = T_{m_i}$). A inserção da mensagem na fila de transmissão pode ocorrer com variabilidade, denotada *jitter* (J). Este pode ser medido como a diferença entre dois instantes de enfileiramento de uma determinada mensagem. O pior caso de tempo de resposta de uma determinada mensagem (R_m) é definido como o mais longo tempo necessário para a mensagem alcançar nó de destino, que é medido em relação ao instante de liberação da tarefa emissora. O tempo de transmissão de uma determinada mensagem m é denominado C_m. Atributos no escalonamento holístico ET tem uma dependência mútua, como por exemplo: considerando a Figura 22 o *jitter* de liberação de c, a depende da chegada de m, que por sua vez depende da interferência de mensagens com prioridade elevada e também do *jitter* de liberação da tarefa s.

O tempo global de sistema estabelece uma sincronização temporal entre nós que é útil para disparo de tarefas em um SDTR de forma sincronizada. Assim, um *offset* pode ser realizado relativo a uma referência comum conhecida por todos os nós presentes na rede. Geralmente, o tempo global é alcançado através de uma sincronização de relógio, mas também pode ser realizado através do envio de uma mensagem específica de referência,

em cada ciclo, sem valor de relógio, a fim de sincronizar todos nós em cada ciclo. Esta última é encontrado nos protocolos FTT, que será discutido mais adiante.

Analisando-se a transação da Figura 22 como um exemplo de um sistema e controle via rede, é necessário observar algumas características em relação à função de controle que será utilizado daqui por diante: a amostragem, a comunicação e a atuação partilham o mesmo período, T; sendo a tarefa de amostragem executada no início do período e, consequentemente, a atuação é realizada no final denotando o atraso de controle, δ; a variação no atraso de controle é denominado $\Delta\delta$, e os desvios nominais do tempo de entrada e de saída são denotadas *jitter*, J_{in} e J_{out}, respectivamente.

4.1.1 Comunicação e processamento por Prioridade Fixa

Considerando a Figura 22 como um SDTR baseado em escalonamento por prioridades fixas, tanto para o escalonamento das tarefas a serem executadas quanto para as mensagens a serem transmitidas via a rede de comunicação, pode-se medir o tempo de resposta de uma amostragem, controle e a atuação. Como um barramento de comunicação baseado em prioridade fixa podemos considerar a rede CAN. A análise do desempenho temporal de SDTR com o uso de escalonamento por prioridade fixa tem sido tema de diversos estudos descritos na literatura (vide por exemplo (TINDELL; CLARK, 1994)). De acordo com o (TINDELL; BURNS; WELLINGS, 1995) uma mensagem pode ser considerada um tipo especial de tarefas, com a diferença principal que mensagens sempre são não-preemptivas. Em sistemas de comunicação baseados no protocolo CAN, mensagens de menor prioridade podem bloquear o envio de uma de maior prioridade, desde que a mensagem de menor prioridade acesse o meio físico antes da mensagem de maior prioridade. A Figura 23 apresenta um diagrama temporal para a execução das tarefas e para a comunicação do sistema de controle descrito na Figura 22 segundo escalonamento por prioridade fixa. As bordas escuras da mensagem m e da tarefa c, a; representa a variação no tempo de transmissão e execução, o que é gerado, respectivamente, pelo método *bit stuffing* do protocolo CAN e por cálculo de ponto flutuante no processo de controle (c), por exemplo.

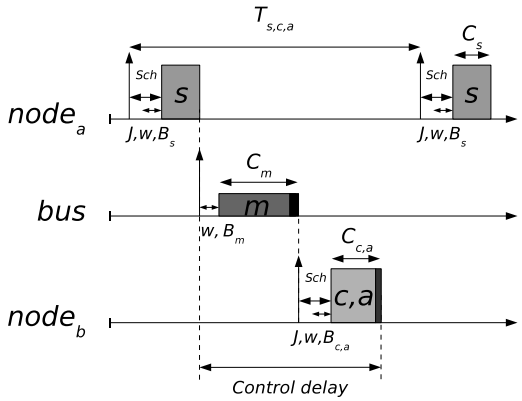

Figura 23: Comunicação e processamento por FP

Para medir o tempo de resposta da transação da Figura 23, em primeiro lugar, é necessário verificar o tempo de resposta da tarefa emissora, R_s:

$$R_s = J_s + C_s + B_s + w_s$$

Onde J_s e C_s são respectivamente o *jitter* de libertação e tempo de execução da tarefa s. B_s é o fator de bloqueo da tarefa s considerando um escalonador não-preemptivo, que é medido como:

$$B = \max_{\forall k \in lp(t_i)} C_k$$

w_s é o tempo no pior caso em que a tarefa s pode esperar na fila, também conhecida como interferência pelas tarefas mais prioritárias, que é medida como:

$$w_s^{n+1} = \sum_{\forall j \in hp(s)} \lceil \frac{w_s^n + J_j}{T_j} \rceil C_j$$

Onde $hp(s)$ é o conjunto de tarefas mais prioritárias do que s. O termo no somatório exprime quantas vezes que cada uma tarefa de prioridade elevada irá interferir com a tarefa s multiplicado com o tempo de execução da tarefa altamente prioritária, C_j. $lp(s)$ é o conjunto de tarefas com prioridade mais baixa do que tarefa s. Se considerar que $node_a$ tem uma conjunto de tarefa n, então é necessário conhecer o tempo de resposta para todas tarefas, a fim de verificar se os seus *deadlines* estão sendo satisfeitos. Em outras palavras, é necessário verificar se o conjunto tarefa é viável.

O próximo passo é conhecer o tempo de resposta do messagem m. Como ela herda o período da tarefa s e, consequentemente, a seu *jitter* de libertação, seu tempo de resposta podem ser mensurados como:

$$\begin{aligned} R_m &= R_s^{max} + C_m + B_m + w_m \\ w_m^{n+1} &= \sum_{\forall j \in hp(m)} \lceil \frac{w_m^n + J_j + \tau_{bit}}{T_j} \rceil C_j \end{aligned}$$

Onde, τ_{bit} é o tempo de transmissão de um bit sobre o barramento CAN (ou tempo de bit). Este termo é derivado da taxa de transferência do protocolo usada no momento.

O último passo desta operação é o tempo de resposta da tarefa receptora, c, a, que é medido da mesma forma que para s no $node_a$.

$$\begin{aligned} R_{c,a} &= R_m^{max} + C_{c,a} + B_{c,a} + w_{c,a} \\ w_{c,a}^{n+1} &= \sum_{\forall j \in hp(c,a)} \lceil \frac{w_{c,a}^n + J_j}{T_j} \rceil C_j \end{aligned}$$

Além disso, c, a herda o *jitter* da mensagem m, que é o pior caso do tempo de resposta R_m^{max}.

Com esta análise, é possível medir o atraso de comunicação fim-a-fim da transação da Figura 23.

O melhor caso para o atraso de controle:

$$\delta_{mim} = C_m + C_{c,a}$$

O pior caso do atraso de controle:

$$\delta_{max} = R_m + R_{c,a}$$

A variação máxima no atraso de controle:

$$\Delta\delta = B_m + w_m + B_{c,a} + w_{c,a}$$

O *jitter* de entrada:

$$J_{in} = B_s + w_s$$

O *jitter* de saída:

$$J_{out} = B_s + w_s + B_m + w_m + B_{c,a} + w_{c,a}$$

4.1.1.1 Ausência de tempo global

Para reduzir o impacto do *jitter* da atividade de controle sobre a atividade de atuação, eles são considerados como duas tarefas independentes. Uma para controle e outro para a atuação. Isto é alcançado através da introdução de um deslocamento no tempo do instante de liberação da tarefa de atuação, denominado *offset*, O, em relação a um ponto comum de referência.

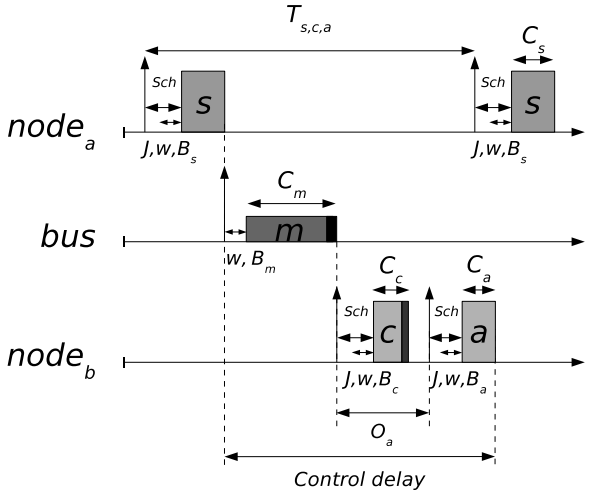

Figura 24: Comunicação e processamento por FP considerando offset

Assim, presume-se que a tarefa de atuação seja liberada seguindo como referência o instante de chegada da mensagem, m, para evitar *jitter* induzido pela tarefa c devido ao

seu tempo de execução variável (Figura 25). Contudo, o *offset* O_{ma} sofre impacto causado pelo *bit stuffing* da mensagem que aumenta o *jitter* da tarefa de controle e consequentemente a tarefa de atuação. As seguintes medidas relacionadas com a função de controle podem ser obtidas:

$$\begin{aligned}
\delta_{mim} &= C_m + O_{ma} + C_a \\
\delta_{max} &= R_m + O_{ma} + R_a \\
\Delta\delta &= B_m + w_m + B_a + w_a \\
J_{in} &= B_s + w_s \\
J_{out} &= B_s + w_s + B_m + w_m + B_a + w_a
\end{aligned}$$

4.1.1.2 Com tempo global

Agora, considerando um tempo global de sistema, a tarefa de atuação pode ser liberada depois que a tarefa de amostragem, a mensagem e comunicação e a tarefa de computação estão concluídos.

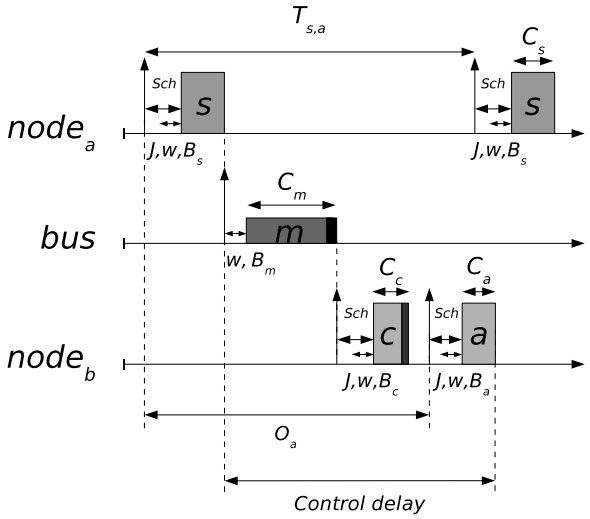

Figura 25: Comunicação e processamento por FP considerando templo global

Neste caso, o tempo de liberação da tarefa de atuação tem um *offset*, O_a, relacionado com a liberação da tarefa de amostragem, reduzindo assim o *jitter* induzido da tarefa de amostragem e mensagem na tarefa de atuação. Além disso, o impacto da variação do tempo de transmissão da mensagem (gerado pelo *bit stuffing*) é eliminado. O atraso de controle sofre impacto do tempo de resposta das tarefas de amostragem e atuação.

$$\begin{aligned}
\delta_{mim} &= O_a + C_a - (R_s) \\
\delta_{max} &= O_a + R_a - C_s \\
\Delta\delta &= B_s + w_s + B_a + w_a \\
J_{in} &= B_s + w_s \\
J_{out} &= B_a + w_a
\end{aligned}$$

4.1.2 Comunicação por Cíclico Estático e processamento por Prioridade Fixa

4.1.2.1 Ausência de tempo global

Nesta condição, o barramento de comunicação baseia-se em TDMA. O atraso de controle mínimo ocorre quando tarefa de amostragem completa imediatamente antes do inicio da janela de comunicação TDMA do $node_a$, a mensagem m sofre um mínimo de ocorrências de *bit stuffing* e tarefa de atuação não sofre qualquer bloqueio ou interferência.

O atraso de controle máximo é atingido quando a tarefa de amostragem é concluída pouco após a janela de comunicação do $node_a$ é iniciada (portanto, a mensagem será transmitida na próxima janela TDMA), a mensagem m sofre um máximo de ocorrências de *bit stuffing* e a tarefa de atuação sofre o máximo de interferência e bloqueio. A Figura 26 apresenta o a condição de atraso máximo.

$$
\begin{aligned}
\delta_{mim} &= C_m + O_{ma} + C_a \\
\delta_{max} &= T_{TDMA} + C_m + O_m + R_a \\
\Delta\delta &= T_{TDMA} + B_a + w_a \\
J_{in} &= B_s + w_s \\
J_{out} &= B_s + w_s + T_{TDMA} + B_a + w_a
\end{aligned}
$$

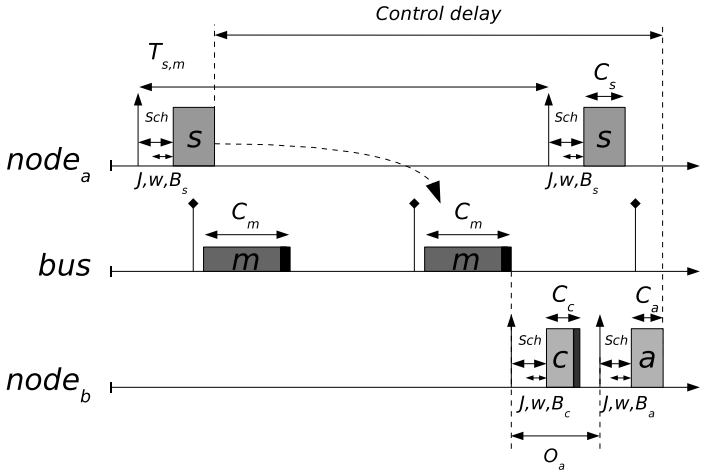

Figura 26: Comunicação por SC e processamento por FP sem templo global

4.1.2.2 Com tempo global

Usando o tempo global de sistema é possível liberar a tarefa de amostragem para terminar antes da respectiva janela TDMA, a fim de garantir a transmissão da mensagem m na sua respectivas janela de tempo. A tarefa de controle é liberada após a chegada da mensagem, assim, sofrendo impacto do *bit stuffing*. A tarefa de atuação, por sua vez, é uma liberada em *offset* em relação a tarefa de amostragem. Atraso de controle mínimos ocorre quando a tarefa de amostragem termina imediatamente antes do início da janela TDMA $node_a$, e a tarefa de atuação não sofre qualquer bloqueio ou interferência. O máximo atraso de controle ocorre quando a tarefa de amostragem termina o mais rapidamente

possível e a tarefa de atuação sofre o máxima de interferência e bloqueando. A variação do tempo de transmissão da mensagem m não prejudica o atraso de controle se, e somente se $O_a \geq R_s + C_m + R_c$.

$$\begin{aligned} \delta_{mim} &= O_a + C_a - R_s \\ \delta_{max} &= O_a + R_a - C_s \\ \Delta\delta &= B_a + w_a + B_s + w_s \\ J_{in} &= B_s + w_s \\ J_{out} &= B_a + w_a \end{aligned}$$

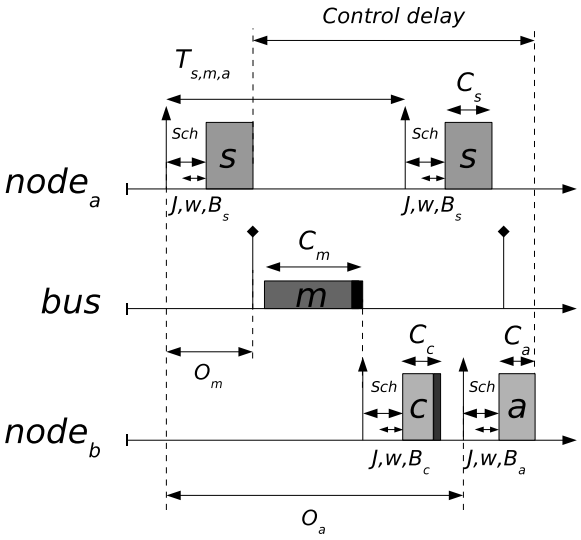

Figura 27: Comunicação por SC e processamento por FP considerando tempo global

4.1.3 Comunicação e processamento por Cíclico Estático

4.1.3.1 Ausência de tempo global

Em um sistema totalmente baseado em escalonamento por SC a ausência do tempo global pode levar a grandes variações no atraso do bloco de controle devido à perda de sincronismo entre nós, o que denota um comportamento assíncrono no barramento de comunicação e no disparo de tarefa. O atraso mínimo de controle ocorre quando a tarefa de amostragem termina imediatamente antes da janela TDMA do $node_a$, e tarefa de computação inicia-se imediatamente após a chegada da mensagem. Atraso máximo ocorre quando a tarefa de amostragem termina depois do começo da janela TDMA do $node_a$, e tarefa de computação inicia antes da que a mensagem chegue com o próximo valor de amostrado, atrasando assim em uma janela TDMA e um período T de unidade (Figura 28). O $\Delta\delta$ é apenas o tempo da janela TDMA e o T devido a falta de sincronismo.

$$\begin{aligned} \delta_{mim} &= C_m + O_{ma} + C_a \\ \delta_{max} &= T_{TDMA} + C_m + T + O_{ma} + C_a \end{aligned}$$

$$
\begin{aligned}
\Delta\delta &= T_{TDMA} + T \\
J_{in} &= 0 \\
J_{out} &= \Delta\delta = T_{TDMA} + T \\
O_{ma} &\geq C_c
\end{aligned}
$$

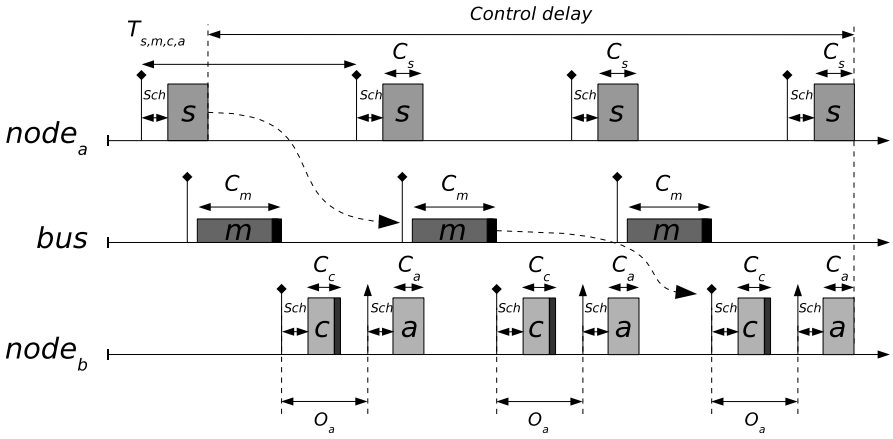

Figura 28: Comunicação e processamento por SC sem tempo global

4.1.3.2 Com tempo global

Em (LONN; AXELSSON, 1999) Lonn cita que com o suporte a tempo global de sistema não existe variação no atraso de controle, uma vez que tarefa de atuação executa em um instante fixo em relação à tarefa de amostragem (*offset*). Mas, para alcançar uma variação nula é preciso um mecanismo de sincronismo sem erros, em seguida, uma variação mínima continuará a existir no momento global. Com o *offset* relatico à tarefa de amostragem é possível eliminar variações na transmissão da mensagem e execução da tarefa geradas pelo método *bit stuffing* na mensagem e variações no cálculos no processo de controle, respectivamente.

$$
\begin{aligned}
\delta_{mim} &= \delta_{max} = O_a + C_a \\
\Delta\delta &= 0 \\
J_{in} &= 0 \\
J_{out} &= 0 \\
O_a &\geq C_s + C_m + C_c
\end{aligned}
$$

4.1.4 Comunicação por Prioridade Fixa e processamento por Cíclico Estático

4.1.4.1 Ausência de tempo global

Neste caso, conforme apresentado na secção 4.1.1, o tempo de resposta para a mensagem pode ser calculado através de análise de escalonamento por prioridade fixada. No entanto, usar escalonamento SC para tarefas não há *jitter* de liberação , assim, sem impacto sobre a mensagem. Sem tempo global o nó não possui qualquer sincronismo entre disparo de tarefas e a comunicação no barramento sobre a política de PF e pode ter grande

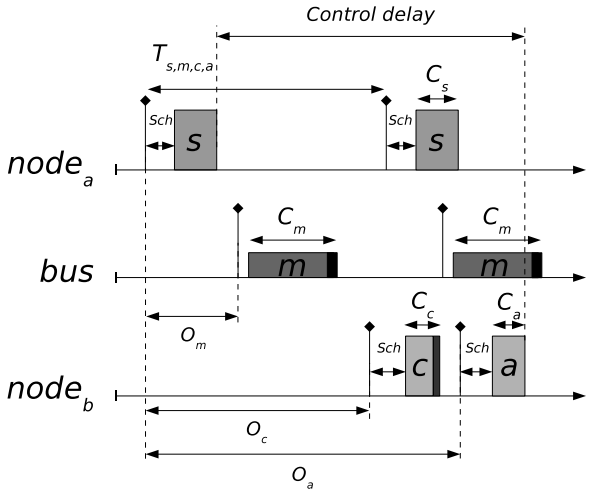

Figura 29: Comunicação e processamento por SC considerando templo global

variação no instante de pico de carga. Assim, a mensagem pode sofrer variação devido ao método *bit stuffing*. O atraso de controle mínimo ocorre quando a mensagem do $node_a$ é enviada sem atraso e a tarefa de controle inicia imediatamente após a chegada da mensagem. O máximo atraso de controle ocorre quando mensagem do $node_a$ sofrer bloqueio máximo, interferência por *bit stuffing*, e tarefa de controle inicia pouco antes da chegada da mensagem devido a perda sincronismo (Figura 30).

$$
\begin{aligned}
\delta_{mim} &= C_m + O_{ma} + C_a \\
\delta_{max} &= R_m + T + O_{ma} + C_a \\
\Delta\delta &= B_m + w_m + T \\
J_{in} &= 0 \\
J_{out} &= B_m + w_m + T \\
O_a &\geq C_c
\end{aligned}
$$

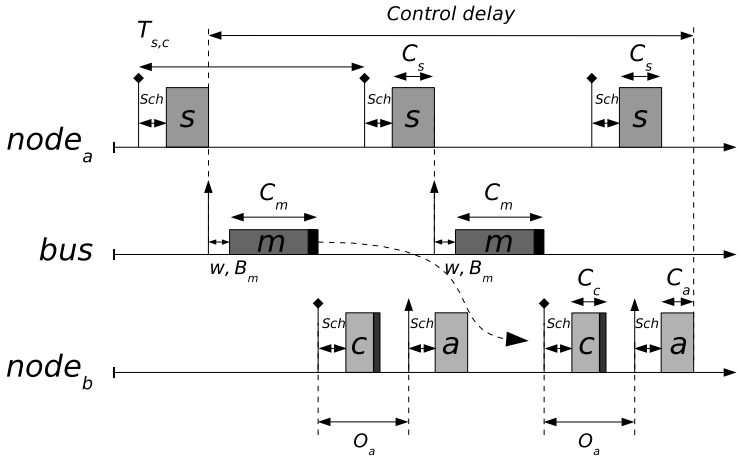

Figura 30: Comunicação por FP e processamento por SC sem tempo global

4.1.4.2 Com tempo global

Utilizando tempo global o resultado pode ser semelhante à condição com escalonamento SC para tarefas mensagens, uma vez que esteja fora de situações de pico de carga no barramento. Como a o escalonamento de mensagem utiliza a política de PF , todas as mensagens do remetente herda o tempo de resposta da tarefa como instante de transmissão. Assim, deve ser garantido um *offset* para a tarefa de controle e atuação tarefa maior do que o tempo de resposta da mensagem.

$$\begin{aligned} \delta_{mim} &= \delta_{max} = O_a + C_a \\ \Delta\delta &= 0 \\ J_{in} &= J_{out} = 0 \\ O_a &\geq C_s + R_m + C_c \end{aligned}$$

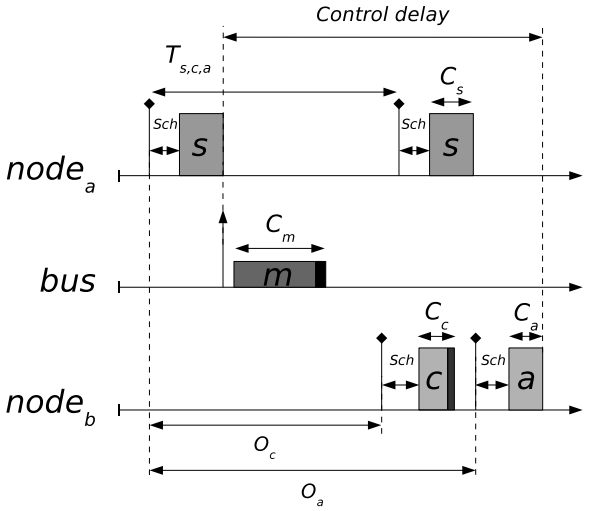

Figura 31: Comunicação por FP e processamento por SC considerando tempo global

4.1.5 Abordagens de tempo global

Conforme apresentado nas secções anteriores, o suporte a tempo global de sistema fornece uma base de tempo comum que torna possível acionar atividade no sistema distribuído de forma sincronizada. E técnicas de tolerância a falha podem ser implementadas para reforçar a segurança do sistema. Em um sistema sincronizado, interações entre diferentes nós podem ser realizadas em instantes fixos no tempo. Atividades inerentes à interação entre tarefas, tal como a preempção com troca de contexto são eliminados, e também bloqueios e interferências são evitados. Na barramento de comunicação, a variação na transmissão gerado pelo método *bit stuffing* também é contornado, porque cada mensagem será transmitida em instantes fixos considerando tal variação no pior caso. Alterações no sistema podem ser realizadas sem efeito negativo no tempo de resposta de tarefas e mensagens, o que significa *composability*. Que é a capacidade de um subsistema ser projetado, testado e está integrado ao sistema sem afetar seu comportamento temporal global. O suporte a tempo global aumenta desempenho e permite uma taxa de amostragem determinística com atrasos mínimos inerentes a variação de relógio e também poupa poder de processamento.

No protocolo TTP /A (EBNER, 1998), uma ciclo de comunicação inicia com uma mensagem especial de referência transmitida pelo nó mestre, denominada *fireworks message*. A chegada da mensagem *fireworks message* nos nós escravos é um evento que inicia uma sincronização para inicio de um ciclo de comunicação. Essa mensagem contém uma identificação o ciclo e ainda pode conter dados. O *fireworks message* pode conter também informações para sincronização de relógio dos nós escravos.

No TTP /C (KOPETZ; BAUER, 2001) a sincronização de relógio estabelece uma base de tempo global sem depender de um servidor central de tempo, por exemplo um nodo mestre com um relógio global. A sincronização de relógio do TTP / C explora o conhecimento comum do plano de envio de mensagens, cada nó mede a diferença entre o tempo previsto e o tempo de chegada real observado de uma mensagem correta para conhecer sobre a diferença entre o relógio do emissor e do receptor. Esta informação é usada por um algoritmo para calcular periodicamente a correção do relógio local, a fim de manter o relógio em sincronia com todos os outros relógios do sistema.

O protocolo FTT-CAN (ALMEIDA; PEDREIRAS; FONSECA, 2002) realiza um sincronismo através de uma mensagem de referência que aciona a transmissão de mensagens em vários nós escravos. O barramento de comunicação é dividido em ciclos EC consecutivos, com duração determinada. Todos os nós são sincronizadas no início de cada EC pela recepção de uma mensagem particular, a TM, enviada pelo nó mestre. Em outras palavras, FTT-CAN usa uma sincronização implícito usando a transmissão regular de uma mensagem de referência, a TM.

O método utilizado no TTP /C pode ser brevemente caracterizada pelo controle de acesso ao meio totalmente distribuído baseado na sincronização de todos nós com um *overhead* de comunicação de serviço de siconização muito baixo. E o método utilizado em TTP /A e FTT-CAN usa controle de acesso centralizado e tem maior *overhead* para gerenciar tal sincronismo. Um aspecto negativo do sincronismo do TTP /C é sua tabela estática de escalonamento, que prejudica a flexibilidade operacional do sistema. Uma vez que o sistema está a ser executado, as mudanças no conjunto de mensagens que têm de ser transmitidas, como a adição de um nova mensagem ou alterar alguns parâmetros de uma já existente, normalmente necessita parar sistema e de carregar uma nova tabela estática.

4.1.6 Conclusões

Esta seção apresentou uma análise entre escalonamento de tarefas e mensagens seguindo as políticas de PF e SC no ponto de vista de sistema de controle via rede. Esta análise revela a abordagem citada na seção 2.1 e também mostra que o melhor caminho é a coexistência de ambas as abordagens, a fim de garantir tanto o atraso de controle (usando política SC) e o tempo de resposta a eventos externos (usando política PF).

O tempo global de sistema pode reduzir o pessimismo causado por bloqueio, interferências e *bit stuffing* em um sistema baseado em escalonamento por SC, considerando que os valores de *offset* são atribuídos de forma adequada. Diferentes métodos foram apresentados nas secções anteriores, cada uma com um foco diferente, quer seja flexibilidade ou menos *overhead*.

4.2 Deficiências no tráfego de mensagens Time-Triggered

Apesar dos problemas identificados afetarem tanto a fase TT como a ET, o foco deste trabalho é sobre o de tempo de resposta na fase TT (tráfego síncrono), porque este exige um elevado grau de previsibilidade quando aplicado em SDTR.

Em SDTR onde a execução tarefa antecede a transmissão de mensagem, qualquer variação na liberação da tarefa e tempo de execução produz atrasos na mensagem, tal como foi apresentado nas seções 2.1 e 4.1. Fontes de variação do tempo de liberação tarefa são por exemplo: preempção por tarefas de alta prioridade, a variação na execução do escalonador ou na latência de interrupção e até mesmo devido variação na própria execução da tarefa. Tais problemas induzem defasagem no tempo na execução de tarefas que é causado pela variação de tempo de resposta das tarefas produtoras de mensagens em sistemas sem qualquer meio de sincronismo. Esta defasagem é denominado como *phasing*. Nolte (NOLTE; HANSSON; NORSTRÖM, 2002) aborda este efeito sobre o tempo de resposta de mensagem em SDTR sobre o protocolo CAN.

Os atrasos também podem ser introduzidos pela estratégia de controle de acesso ao meio de comunicação do protocolo utilizado. Esta questão foi apresentada em 2.1.1.

Outra fonte de variabilidade temporal na comunicação é relacionado com os mecanismos de integridade de dados usados pelos protocolos, tal como o método de *bit stuffing* do protocolo CAN (e, portanto, também presente no FTT-CAN), que aumenta o comprimento da mensagem. O formato de mensagem CAN contém 47 bits de informações de controle do protocolo (o ID, CRC, ACK e bits de sincronização, etc.) A transmissão de mensagem utiliza o *bit stuffing* que insere um bit após cada cinco bits consecutivos de mesmo valor. Os segmentos "start of frame", "identifier", "control field", "data field"and "CRC sequence"são codificadas pelo método *bit stuffing*. Os demais segmentos do frame de dados e remoto (delimitador CRC, ACK, e o *end of frame*) são fixados de forma a não receber os *bit stuffing*. Também os frames de sobrecarga e de erro são fixos e inalterados no instante da transmissão (CIA, 2001b).

O número de *stuff* bits inseridos depende do padrão de bits de uma mensagem em questão, por exemplo: uma mensagem CAN com 8 bytes de dados e 47 bits de controle pode ser transmitida com 0 a 19 *stuff* bits. Isso torna difícil uma análise precisa de tempo de resposta no protocolo CAN, e, por esta razão, algumas trabalhos não consideram esta incidência (TINDELL; BURNS; WELLINGS, 1995).

O impacto do *bit stuffing* aumenta quando a rede possui baixa velocidade de transmissão. Quanto menor for a taxa de transferência da rede, maior será o impacto na ordem de unidades de tempo. Por exemplo, a Figura 32 apresenta uma mensagem de 8 bytes de dados em uma rede CAN de 500kbps.

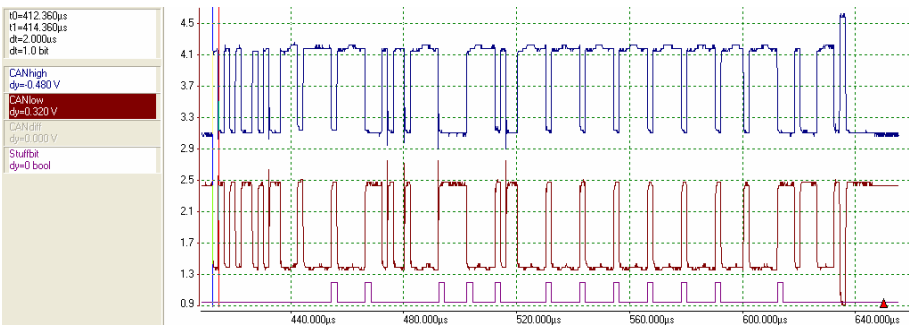

Figura 32: Impacto do *bit stuffing* no tempo de transmissão a 500kbp/s

Como sinalizado na última linha do gráfico a incidência de *stuff* bits nesta mensagem totaliza 12 bits, que por sua vez induzem um atraso de 24μs.

Agora, a Figura 33 apresenta uma mensagem de 8 bytes de dados em uma rede CAN de 50kbps.

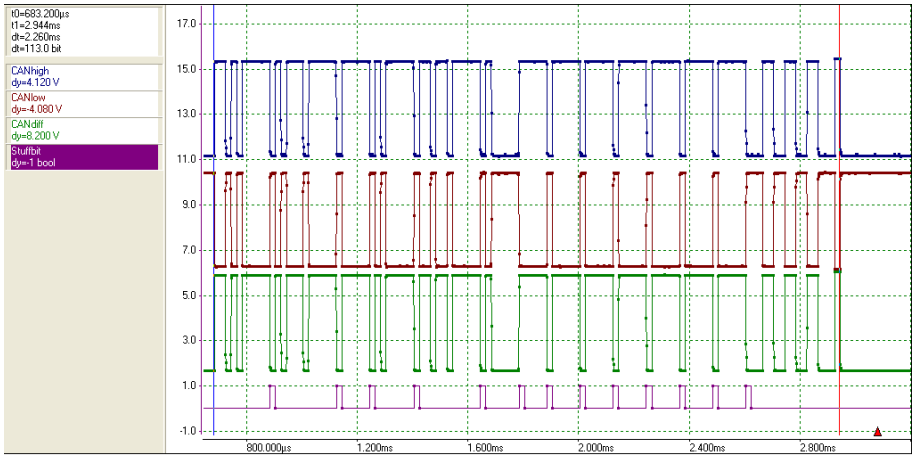

Figura 33: Impacto do *bit stuffing* no tempo de transmissão a 50kbp/s

Nesta mensagem, a incidência foi maior totalizando 13 bits, que por sua vez induzem um atraso de 260μs.

Em (NOLTE; HANSSON; NORSTRÖM, 2003) é apresentado uma análise probabilística de pior caso de tempo de transmissão baseada na utilização da distribuições de *bit stuffing* ao contrario da consideração de um valor de incidência máxima no pior caso. A Figura 34 ilustra este problema no FTT-CAN, considerando uma situação hipotética de uma rede com quatro nós sendo que cada um com uma mensagem TT com prioridades diferentes.

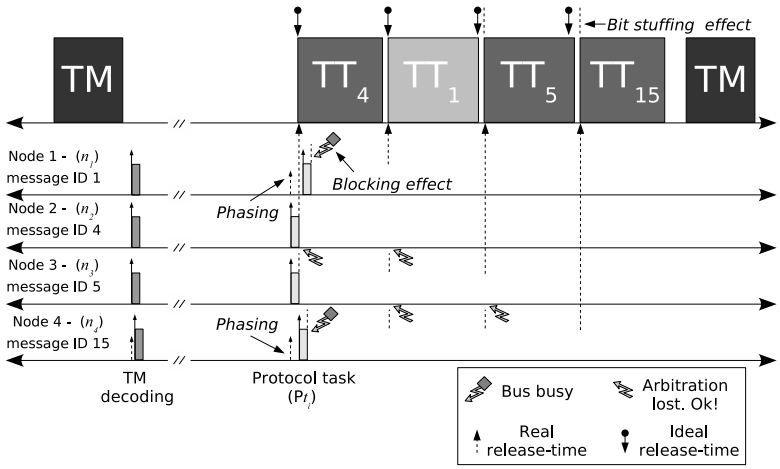

Figura 34: Impacto do *bit stuffing* e efeito bloqueio

O processo de liberação da mensagem, que é responsável pela transmissão das mensagens TT em cada nó pode ser afetado por variações no serviço de interrupção e no escalonador do RTOS (Real Time Operating System) em uso. Esta situação pode gerar uma condição de bloqueio devido ao defasagem temporal do processo de liberação de mensagem, ilustrado na Figura 34. Quando o nó 1 (após uma condição de defasagem temporal) tenta transmitir sua mensagem TT_1 encontra o barramento ocupado com uma mensagem de menor prioridade, a mensagem (TT_4) enviada pelo nó 2. Esse é um típico caso de inversão prioridade no protocolo CAN, que incide no aumento do tempo de resposta. Também, o processo de liberação de message do nó 4 sofre a mesma defasagem temporal, mas sem condição de bloqueio devido a sua prioridade da mensagem. A in-

trodução desta defasagem com duração mínima igual ao tempo de transmissão e um bit na rede CAN é suficiente para causar uma condição bloqueio na primeira mensagem do segmento TT, a qual é de maior prioridade.

Influência do método *bit stuffing* também é representado na Figura 34. Este conduz a uma maior latência que gera um *jitter* na transmissão de mensagem subsequente, pois o instante de transmissão será variável. Quanto maior o número de mensagens a serem transmitidas na fase TT, maior será este impacto. Mensagens escalonadas para transmissão no fim da fase TT sofre um maior *jitter*. Uma vez que esse atraso é medido em bits, quanto menor for a taxa de transmissão maior será o impacto.

4.2.1 Resultados práticos

Para avaliar o protocolo FTT-CAN seguindo sua proposta de implementação original (apresentada em (ALMEIDA; PEDREIRAS; FONSECA, 2002)), um sistema hipotético foi criado para demonstrar a deficiências no tráfego de mensagens Time-Triggered. Este sistema é composto por 3 nós FTT-CAN e 5 mensagens TT distribuído entre os nós.

Node 1 FTT-Master (TM message);

Node 2 FTT-Slave 01 (TT messages 2, and 4);

Node 3 FTT-Slave 02 (TT message 1, 3 and 5);

A fim de observar os inconvenientes acima apresentados, todas as mensagens TT têm mesmo período (5 ms). Em outras palavras, todas as mensagens TT estarão presente em todos os ECs.

As mensagens 1, 2, 3 e 5, foram analisada para apresentar o impacto do *bit stuffing* e o efeito do bloqueio. A Figura 35 apresenta histogramas com a distribuição do *jitter* destas mensagens. A TM não têm grande variação, o *jitter* apresentado é gerado pela variação do manipulador de interrupção do RTAI. A implementação do FTT-Master usa um escalonador estático, onde um vetor com todas as sequências de TT mensagem foi predefinido em fase de projeto. Ou seja, o FTT-Master não possui qualquer carga computacional durante o ciclo EC.

A mensagem TT 1 e 2, devido às suas prioridades, devem ser transmitidas no início da fase TT e podem sofrer o efeito bloqueio devido à defasagem temporal do processo de liberação de mensagem causado pela variação dos serviços de interrupção de cada nó. Seus histogramas apresentam, nas extremidades, grande variações na ordem de -272μs a 268μs gerados por inversão de prioridade devido ao efeito de bloqueio.

O efeito *bit stuffing* pode ser percebido nas mensagens TT 3 e 5, que estão no fim da fase TT. A mensagem TT 5 sofre o maior impacto, pois é a última, a sua variação é de cerca de -28μs a 28μs.

A Figura 36 mostra uma imagem de um osciloscópio apresentado um ciclo EC desta primeira implementação do FTT-CAN. Todas as Mensagens TT são liberadas simultaneamente no início da fase TT. Portanto, todas as mensagens são transmitidos sequencialmente seguindo apenas suas prioridades através do ID no processo de arbitragem. Portanto, somente o *inter-Frame* CAN separam as mensagens.

4.3 Deficiências na execução de tarefas Time-Triggered

Em relação ao escalonamento holístico no protocolo FTT-CAN, os autores em (CALHA; FONSECA, 2002) e (CALHA; SILVA; FONSECA, 2006) não especificam uma janela

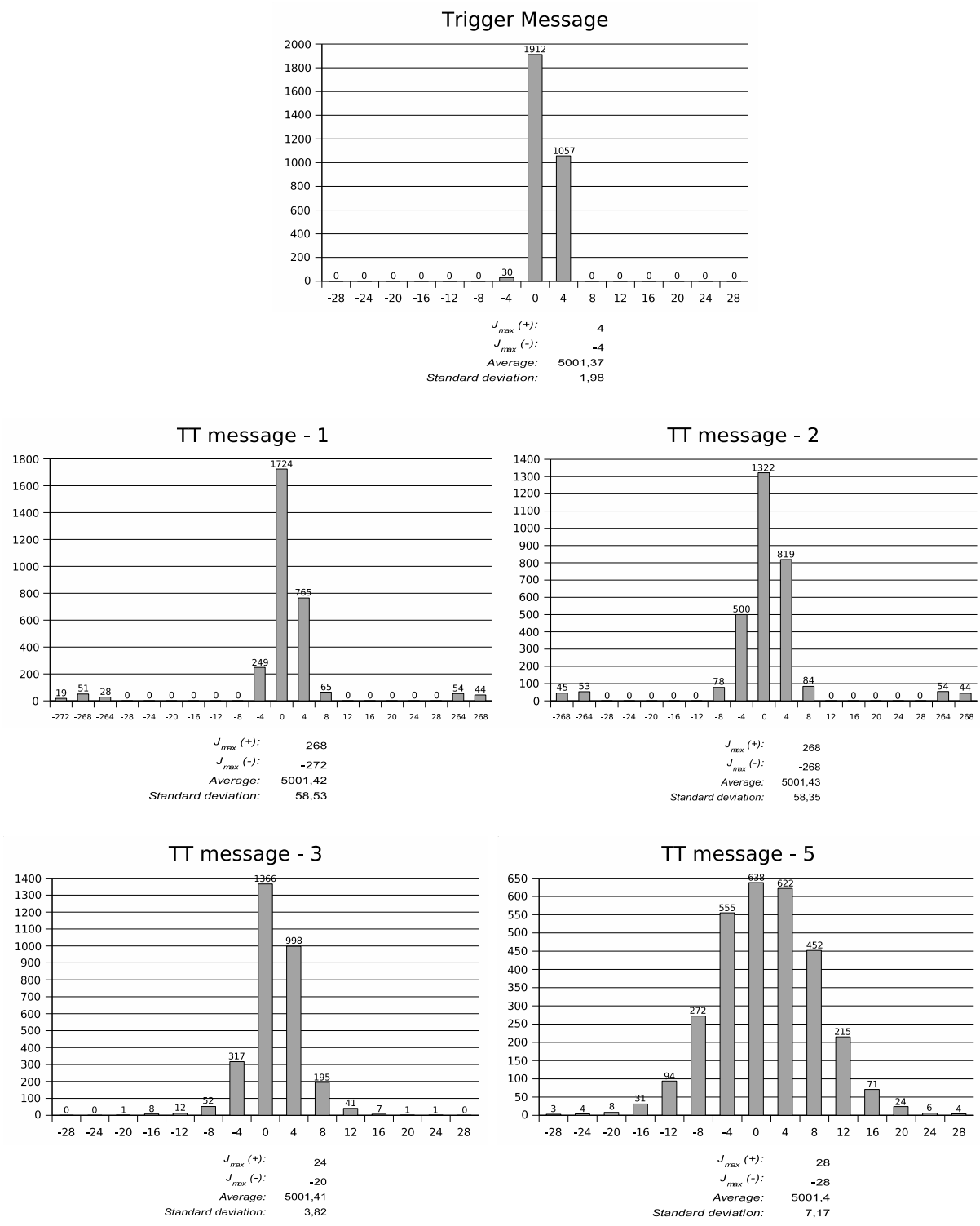

Figura 35: Resultados da implementação original do protocolo FTT-CAN

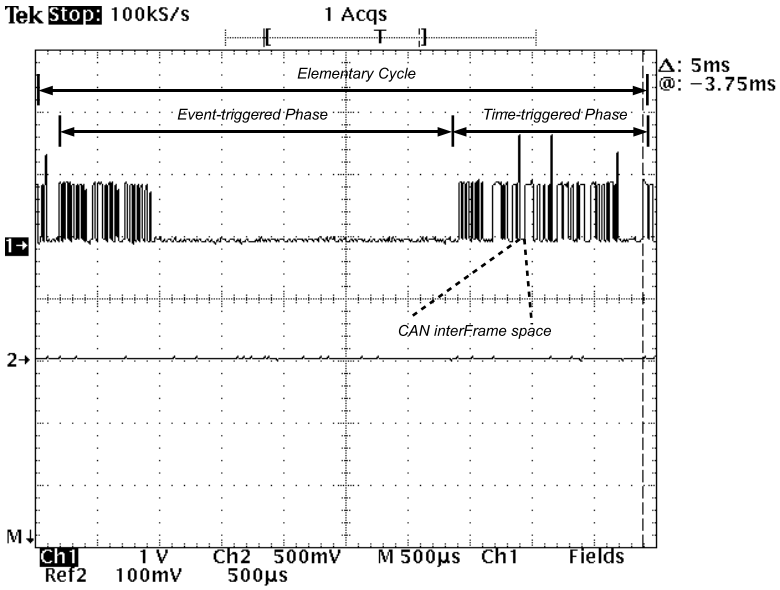

Figura 36: Ciclo Elementar da implementação original do protocolo FTT-CAN

para a de tarefas síncronas (tarefas TT).

Os autores em (CALHA; SILVA; FONSECA, 2006) consideram a janela de execução de uma tarefa síncrono como o intervalo entre o instante de liberação e o *deadline* da tarefa. No entanto, os autores não definem um instante para a liberação de uma determinada tarefa TT em um dado EC. Além disso, os autores em (CALHA; FONSECA, 2002) consideram a janela de transmissão como o intervalo entre o instante de liberação e o *deadline* da mensagem. Mas, neste caso, todas as mensagens TT são liberadas no ponto inicial da fase TT.

De acordo com (CALHA; FONSECA, 2002) uma tarefa TT é liberada assim que a TM é decodificada, onde cada nó verifica se há alguma tarefa para execução no respectivo ciclo EC. Sendo assim, todas as tarefa TT terão a execução na fase ET.

Esta abordagem degrada o tráfego EE com execuções de tarefa não-assíncrona, gerando assim interferências sendo que as tarefa ET têm prioridade menor do que as tarefa TT. Outro problema na utilização desta abordagem é a variação no processo de decodificação da TM, que é inevitável, porque pode ocorrer TMs com mais tarefas/mensagens decodificadas do que outras, por isso o tempo de computação é variável.

Considerando as notas apresentadas acima, a Figura 37 apresenta um exemplo desta abordagem.

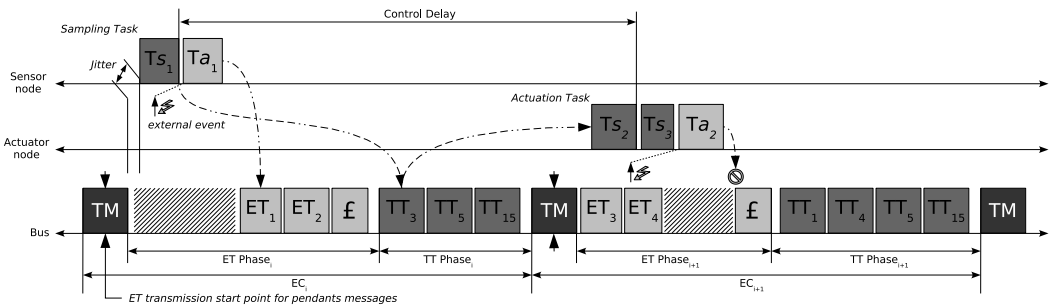

Figura 37: Disparo de tarefas no FTT-CAN

Como pode ser visto, tarefa TT executando fora de sua respectiva fase lesa a execução de tarefas ET. O tempo de execução de uma tarefa TT irá intervir no tempo de resposta a eventos externos que podem ocorrer durante o tempo de sua execução. Afinal, o fase ET foi destinada à serviços assíncronos. Por exemplo, no EC_i da Figura 37, não existe qualquer mensagem ET pendente para transmissão, por isso a mensagem ET_1 é transmitida somente após a tarefa Ts_1 termine sua execução e a tarefa Ta_1 processe seu respectivo evento externo que ocorreu durante a execução de Ts_1. No próximo EC, EC_{i+1}, existem duas mensagens ET (ET_3 e ET_4) pendentes desde a última EC, que são transmitidas corretamente em sua fase sem interferências. Porque, na FTT-CAN todas as mensagens ET pendentes são liberadas no meio de transmissão da mensagem TM como ilustrado na Figura 37. Deste modo, somente após as tarefas Ts_2 and Ts_3 terminarem sua execução a tarefa ET Ta_2, resultante de outro evento externo, poderá ser executada. No entanto, após a execução de Ta_2 não há tempo para a transmissão da mensagem ET resultante da tarefa, e esta será colocado em uma fila de espera para o próximo ciclo EC.

Consequentemente, com estas evidências, podemos concluir que executar tarefas TT na fase ET pode gerar interferências nas tarefas e mensagens ET, provocando um *dead time* nesta fase (sombra na área da Figura 37).

4.4 Abordagem proposta para a fase Time-Triggered

4.4.1 Tráfego de mensagens Time-Triggered

Para superar os inconvenientes apresentados na transmissão de mensagem TT, um método baseado em *offset* é apresentado para forçar a ordenação correta das mensagens TT de forma a reduzir o *jitter* inerente do efeito bloqueio e do método *bit stuffing*. O trabalho original sobre este método *offset* foi realizado por Liu e Sun (SUN; LIU, 1996), que é um protocolo de sincronização para garantir a relação de precedência entre as tarefas periódicas pela a inclusão de deslocamentos no tempo (*offsets*).

Atribuir um *offset* para toda tarefa de protocolo - responsável pela liberação das mensagens TT em cada EC -,($O_{P_{t_i}}$), de modo que sua liberação será sempre superior ao pior caso do tempo de transmissão do conjunto de mensagens TT (C_{TT_m}), com isso reduzindo pessimismo gerado pelo *bit stuffing* e garantindo a eliminação do efeito de bloqueio na fase TT do protocolo FTT-CAN.

É necessário considerar o pior caso de ocorrências de *bit stuffing* e incluir o espeço *inter-frame* para calculo do C_{TT_m}. O offset de sistema (O_{sys}) é um valor fixo de cerca de poucos bits, que é ajustado em tempo de projeto como também o C_{TT_m}. O O_{sys} tem a finalidade de assegurar uma lacuna temporal entre o final de uma mensagem TT o início de outra. Ista lacuna varia devido as ocorrências *bit stuffing*. O cálculo de *offset* para cada mensagem TT é dado pela equação:

$$O_{P_{t_i}} = (C_{TT_m} + O_{sys}) m_{indexEC}$$

Onde $m_{indexEC}$ é a posição da mensagem no respectivo ciclo EC que está em conformidade com a prioridade, e este deve ser ≥ 0. Assim, a primeira mensagem no respectivo EC terá $O_{P_{t_{i=0}}} = 0$.

Ao usar este método cada tarefa de protocolo terá um deslocamento O_P no tempo, promovendo a liberação de mensagem num instante de barramento ocioso - após a transmissão da mensagem anterior. Além disso, este método torna possível a utilização de mensagens TT com tamanho de dados diferentes, o que não era possível na implementação original do protocolo FTT-CAN.

4.4.2 Resultados Práticos

Para avaliar e comparar a nova proposta de tráfego TT com a primeira implementação - apresentada na seção 4.4.1 -, o mesmo sistema hipotético foi utilizado.

A Figura 38 apresenta histogramas do conjunto de mensagens apresentadas na seção 4.4.2. Como na primeira implementação, a TM não tem grande *jitter*.

As mensagem TT 1 e 2 não sofrem inversão de prioridade devida ao efeito de bloqueio, que foi completamente eliminado com a nova abordagem com *offset*. Agora, a sua variabilidade é de cerca de -4μs a 8μs gerado pelo *jitter* do serviço de interrupção do RTAI.

O efeito *bit stuffing* foi reduzido conforme pode ser visto no histograma das mensagens TT 3 e 5. A variação é de cerca de -16μs to 16μs.

A Figura 39 mostra a imagem de um osciloscópio com um ciclo EC da implementação da nova proposta com *offset*. Todas as Mensagens TT são liberadas em sua janela de tempo específica na fase TT de acordo com sua prioridade. Todos os *offsets* são calculados dinamicamente em cada ciclo EC seguindo a quantidade de mensagens TT a serem transmitidas e suas respectivas prioridades.

4.4.3 Execução de tarefas Time-Triggered

É notório que interferências na fase ET na implementação original do FTT-CAN podem ser geradas através da execução de tarefas TT. Para tentar ultrapassar este problema, uma proposta de janelas de execução é proposta. O objetivo é restringir a execução de tarefas TT dentro de sua respectiva fase ou o mais próximo possível, reduzindo as interferências na fase ET. Assim, um tarefa TT é liberada dentro de uma janela de tarefa definida. A duração dessa janela de tempo é fixa e definida considerando os valores máximos WCET do conjunto de tarefa TT, adicionados a um fator *jitter*. Este fator deve ser o pior caso do *jitter* de execução da tarefa e do serviço de interrupção. A janela da tarefa produtora sempre antecede a janela da mensagem a ser produzida, e a janela de uma tarefa consumidora é sempre realizada no próximo EC da janela da mensagem a ser consumida. A Figura 40 mostra esta abordagem proposta.

A janela de tarefa da primeira janela de mensagem a ser produzida na fase TT do ciclo EC será lançada no final parte da fase ET. É preciso ter em conta mensagens com o mesmo período do mesmo nó, para isto deve ter $ECperiod = minorPeriodOfMessage/2$.

Desta forma, não se pode aplicar deslocamentos entre mensagens TT em diferentes EC permitindo que a janela da tarefa produtora preceder a janela da mensagem. A atribuição das janelas de tarefas é feito *on-line* durante cada EC enquanto o processo de decodificação da TM esta em curso.

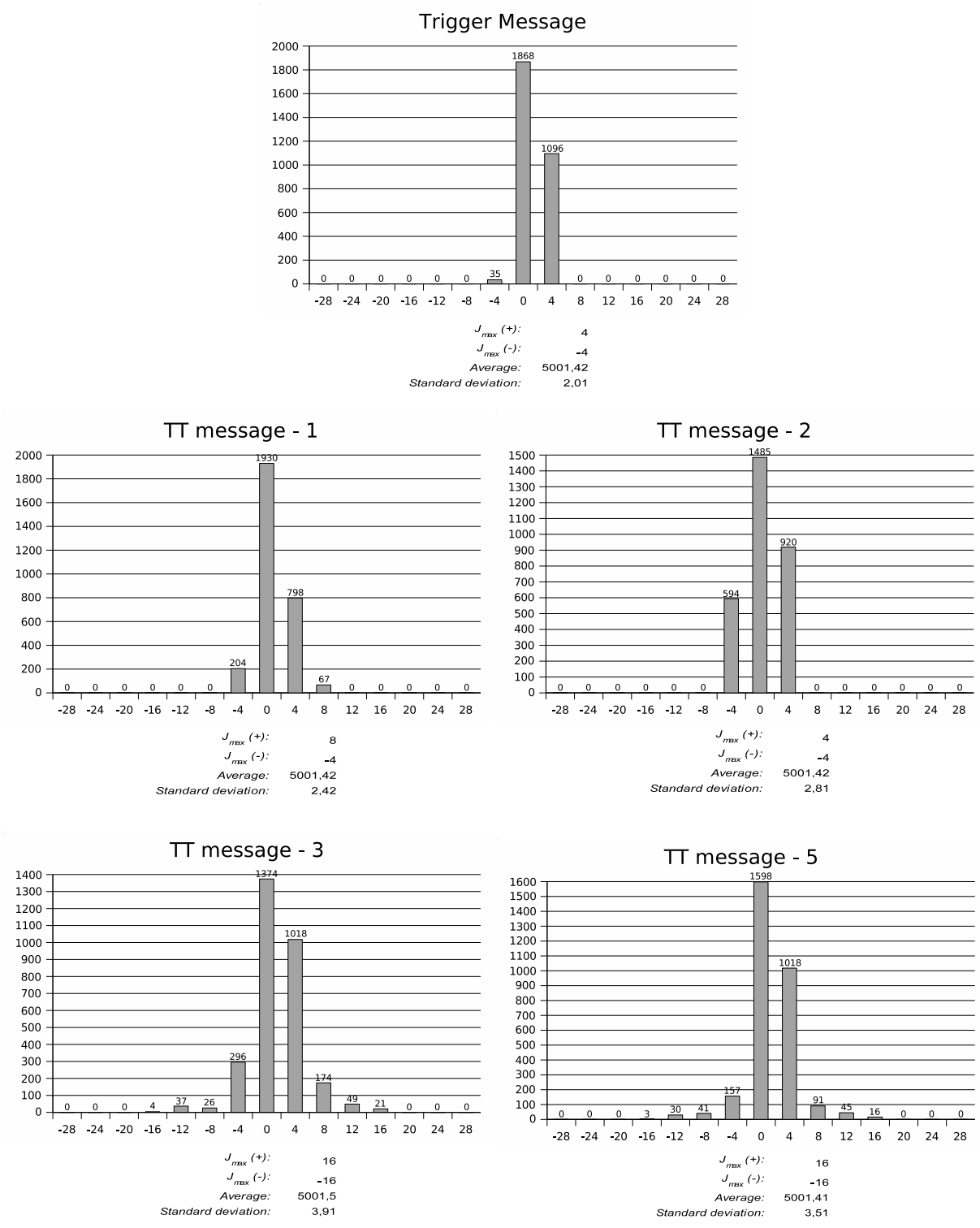

Figura 38: Nova abordagem para o protocolo FTT-CAN

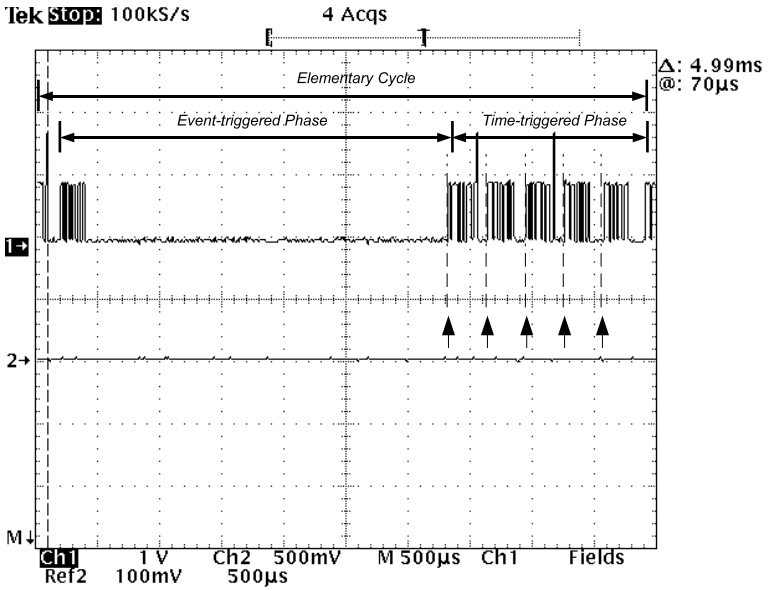

Figura 39: Resultado da implementação da nova abordagem

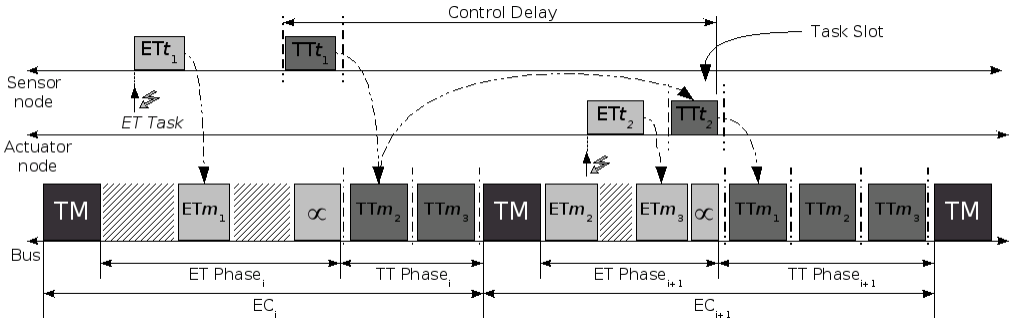

Figura 40: Nova abordagem para disparo de tarefas

5 IMPLEMENTAÇÃO E VALIDAÇÃO DAS PROPOSTAS

Este capítulo descreve alguns resultados práticos das abordagens propostas para protocolo FTT-CAN aplicadas em um estudo de caso em uma aplicação automotiva.

5.1 Implementação do sistema

A utilização de um RTOS (Real-Time Operating System) é obrigatório em certas circunstâncias, quando vários fluxos de instruções deve ser multiplexados para execução em um único processador com garantias de tempo. Dependendo da complexidade da aplicação, a utilização de um único fluxo de execução para acomodar todas as funções de sistema pode resultar em um código não muito claro, complexo e com alto custo de desenvolvimento. Um RTOS fornece várias vantagens no ponto de vista de implementação da aplicação, tais como código de tamanho reduzido, menor complexidade e facilidade na manutenção e reutilização.

Como resultado de uma análise comparativa de RTOSs, a escolha foi adotar o μClinux[1] (DIONNE; DURRANT, 2002). Uma vantagem dos sistemas operacionais baseados em Linux é sua arquitetura modular do kernel que aumenta a flexibilidade do sistema em tempo de execução. Em outras palavras, os módulos do kernel podem ser carregados dinamicamente. E considerando uma situação particular, por exemplo, um driver de dispositivo Ethernet e pilha de protocolo podem ser carregados quando for necessário, e removidos quando não há necessidade de uso.

Infelizmente, o μClinux por si só não é um RTOS. Seu kernel não é preemptivo e seu escalonador não fornece garantias temporais. No entanto, atualmente, existem alguns projetos com o objetivo de fornecer extensões de tempo real para o Linux, e as mais relevante delas são o Real-Time-Linux (YODAIKEN; BARABANOV, 2003) e os RTAI (MANTEGAZZA, 2001) (Real-Time Application Interface). Discussão sobre estas extensões não é o foco deste trabalho. O leitor é referenciado para (ANDERSSON; LINDSKOV, 2003) que apresenta um estudo comparativo entre a RT-Linux e RTAI. Para o propósito deste trabalho a extensão RTAI foi considerada a melhor opção. Os fatores que consolidaram a escolha foram: 1) é um projeto *open source*; 2) tem um código de base maduro; 3) existem ferramentas gratuitas disponíveis, 4) pequeno tamanho após compilado (*small footprints*); 5) comunidade de desenvolvimento ativa e 6) suporta uma ampla variedade de arquiteturas tal como x86, PowerPC, ARM, MIPS e ColdFire. O μClinux juntamente com o RTAI apresentam características de um RTOS que foram apresentados anteriormente.

[1] Sistema operacional Linux para aplicações embarcadas, baseado em processadores sem unidade de gerenciamento de memoria (Memory Management Unit)

5.1.1 μClinux - Visão Geral

O μClinux ou Linux para micro-controlador é uma variante do sistema operacional GNU Linux. Foi iniciado em 1997 com objetivo de criar uma nova versão variante do kernel do Linux 2,0 para micro controladores de baixo custo, que em sua maioria não possui unidade de gerenciamento de memória (MMU). Isto significa que as diferentes aplicações desenvolvidas sob o GNU Linux também podem ser adaptadas para suportar versão sem MMU do Linux, geralmente com pequenas mudanças. Atualmente o kernel suporta múltiplas plataformas, incluindo Coldfire, Axis ETRAX, ARM, Atari 68k e outros.

μClinux foi especialmente concebido para a aplicação baseada em microprocessadores embutidos sem MMU. A ausência da MMU é um fator de redução de custos de microprocessadores. Ainda no ponto de vista do custo dos componentes de hardware, dispositivos de memória têm contribuição significativa sobre o projeto de hardware. Por isso, μClinux possuie pequeno tamanho pós-compilação (cerca de 400 a 900 kB), que permiti desenvolver aplicações com tamanho de memória limitado. Também possui suporte a rede de comunicação, incluindo uma completa pilha TCP /IP e suporta uma ampla gama de diferentes protocolos de rede. Além disso, o μClinux suporta diferentes sistemas de arquivos.

Como limitação, ele não oferece proteção de memória e a memória virtual é toda memória física. Assim qualquer programa pode, por motivos de falhas de implementação, acessar região protegida de memória.

O μClinux usa uma versão leve da biblioteca padrão C, denominados μClibc. Ela é baseado na Biblioteca Linux C-8086, reduzida para um pacote compacto. O principal objetivo era conceber um espaço otimizado para a biblioteca C para microcontroladores sem MMU como o Dragonball, Coldfire e alguns ARMs. Ele também estabelece um padrão de APIs Linux libc, assim os desenvolvedores podem migrar aplicativos a partir de sistemas operacionais baseados em POSIX para o μClinux.

Como no ambiente Linux, o software de aplicação no μClinux se reside no espaço de usuário e não possui permissão de acesso ao hardware, o que é feito através de drivers específicos.

O kernel do μClinux pertence a licença GNU GPL1[1], como todos os aplicativos provenientes do pacote de distribuição.

5.1.2 RTAI - Visão Geral

RTAI é o acrônimo de *Real-time Application Interface*. Trata-se de resultados de pesquisas sobre os sistemas de controle de tempo real, iniciada em 1999 pelo DIAPM (Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano). RTAI é um projeto *open source* com uma comunidade de desenvolvimento ativa.

O conceito básico do RTAI é muito semelhante ao do RTLinux. Uma nova camada de software, chamado HAL (*Hardware Abstraction Layer*), foi introduzida entre o kernel do Linux e do hardware com total controle de interrupções e principais recursos do processador. A HAL fornece uma interface para o hardware, tanto para o Linux como para o núcleo de tempo real. Esta capacidade com a HAL é usado em outros sistemas operacionais, tais como RTLinux e Ecos. Cerca de 100 linhas de código é tudo que é alterado ou adicionado no kernel do Linux após da realização do *patch* com código RTAI. Claramente, o conceito RTAI HAL facilita e simplifica o suporte do RTAI. O núcleo do RTAI oferecer as características de tempo real para escalonamento de tarefas e serviços de

[1]GNU General Public License, see http://www.gnu.org/copyleft

interrupção. Estes recursos não são diferentes de outros sistemas operacionais de tempo real.

Uma característica particular do RTAI é o componente LX /RT, que torna características de *soft* e *hard real-time* disponíveis ao espaço de usuário no Linux. RTAI oferece uma API simétrica para o espaço de kernel e de usuário. As mesmas funções de tempo-real são utilizáveis com a mesma chamadas de função no nível de espaço de usuário, bem como no espaço de kernel. Além disso, o IPC (*inter-process comunicação*) que a LX /RT oferece entre o espaço de usuário e kernel trabalha com uma API simétrica.

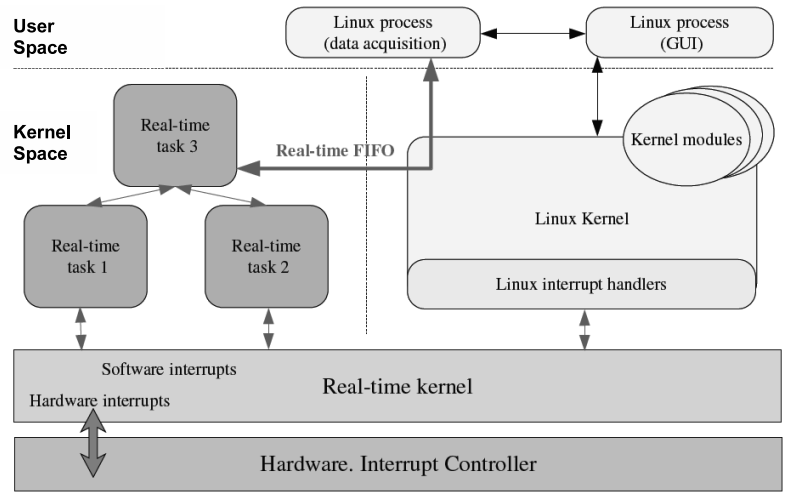

Figura 41: Arquitetura de software RTAI (MANTEGAZZA, 2001)

O escalonador do RTAI trata o kernel do sistema operacional Linux como a tarefa ociosa (*Idle task*). Processos do Linux somente são executados quando não existem tarefas de tempo real para execução ou quando kernel de tempo real está inativo. O processos do Linux nunca podem bloquear interrupções ou evitar preempção.

Além disso, o núcleo é estendido com alguns pacotes extras, como por exemplo: comunicação entre processos, drivers para protocolos de rede e linha serial; interface POSIX; interfaces para LabVIEW, Comedi e Real-time Workshop, etc.

O projeto foi portado para uma grande variedade de arquiteturas, tais como:

- x86;
- PowerPC;
- ARM (StrongARM; ARM7: clps711xfamily, Cirrus Logic EP7xxx, CS89712, PXA25x);
- MIPS;
- ColdFire 5282;
- CRIS

RTAI foi implementados através de módulos de kernel do Linux oferecendo uma boa modularidade. Os principais módulos são:

rtai módulo núcleo;

rtai_sched escalonadores;

rtai_fifos fifos para comunicação entre aplicação de tempo-real e de usuário;

rtai_shm memória compartilhada entre aplicação de tempo-real e de usuário;

lxrt suporte a escalonamento de tarefas no nível de aplicação de usuário;

rtai_pqueue, rtai_pthread, rtai_utils módulos POSIX RTAI;

A documentação do RTAI está bem escrita e disponível. RTAI é um projeto liberado sob licença LGPL[1].

5.2 Protocolo FTT-CAN no RTAI/μClinux

Como a extensão RTAI fornece meios para adicionar suporte a recursos de tempo real no kernel do μClinux, a pilha do protocolo FTT-CAN e aplicativos de software foram incluídas no de espaço de kernel objetivando garantir previsibilidade na execução das tarefas. Do mesmo modo, sobre o espaço de usuário decidiu-se atribuir tarefas que não tenham restrições temporais, como por exemplo, processo de diagnóstico. A arquitetura de software do FTT-CAN sobre RTAI é apresentada na Figura 42, no qual as setas demonstram as relações entre os componentes, onde os módulos RTAI são indicados em cor cinza e os módulos desenvolvidos são indicados em cor branca.

No lado esquerdo a pilha do protocolo é apresentada, todos os módulos são descritos a seguir:

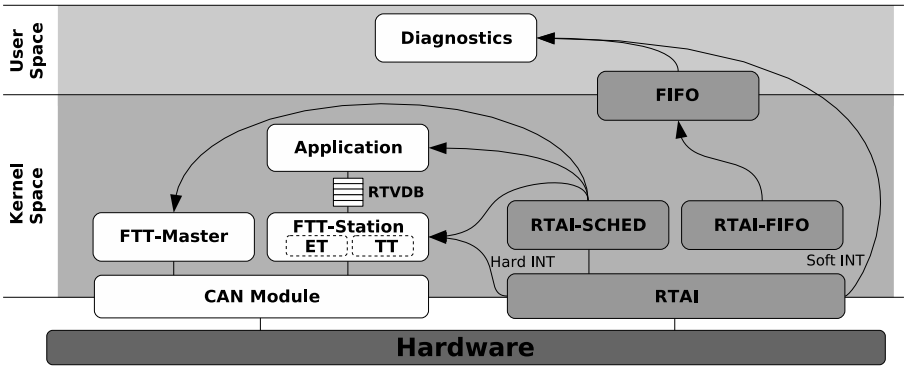

Figura 42: Arquitetura de software do FTT-CAN com RTAI

1. No topo está o módulo `Application` que representa o aplicativo com as tarefas de tempo real do sistema.

2. Na base da pilha FTT-CAN, está o `CAN Module` que fornece alguns métodos de acesso ao controlador CAN e registos internos. Decidiu-se construir primitivas dedicadas com objetivo de diminuir o *overhead* que se teria caso fosse adotado um driver COTS (*commercial-off-the-shelf*) existente, como por exemplo o driver *CAN4Linux*. O Módulo CAN consiste apenas em um conjunto de primitivas de acesso ao controlador CAN para os módulos *FTT-Station* e *FTT-Master*. Em outras palavras, este módulo pode ser nomeado como um Hardware Abstraction Layer.

[1] *GNU Lesser General Public License*, ver em http://www.gnu.org/copyleft/lesser.html

3. O módulo *FTT-Station* é implementado com tarefas de alta prioridade (ET e TT apresentada na Figura 42) e são responsáveis por todas as mensagens ET e TT trocadas sobre o sistema. Por exemplo, ele acessa a base de dados, quer atualizando com novos dados a partir da rede ou para carregar um dado para transmissão dentro de uma mensagem ET ou TT. O serviço de interrupção do RTAI é o recurso principal deste módulo. Cada vez que um novo ciclo EC é iniciado pela transmissão da TM, uma interrupção é ativada acionando a execução do módulo `FTT-Station`. Posteriormente, o conteúdo da TM é decodificado pelo módulo para conhecer quais *Real-Time Variable* (RTV) no nível da aplicação devem ser enviadas ou recebidas - produzidas ou comsumidas - e quais *Real-Time Task* (RTT) devem ser liberadas na próxima fase TT do ciclo EC.

   Um RTV é qualquer variável de aplicação cuja a validade é restrita no tempo. A estrutura de dados de uma RTV tem os seguintes atributos:

   - Identificador
   - Tamanho de dados
   - Apontador para *buffer* de dados
   - Flag de indicação dado produzido ou consumido
   - Validade do dado no tempo
   - Apontador para a respectiva tarefa de tempo-real
   - Flag de indicação variável TT ou ET

   O conjunto de RTVs de interesse em um nó é alocado no *Real-Time Variable Data Base* (RTVDB), representando a interface entre aplicação e a pilha de protocolo (Figura 42). Pode-se compara-lo com a CNI (*Communication Network Interface*) em outros protocolos, como o TTP /C e TTP /A. A criação do RTVDB na memória é realizada durante a fase de *start-up*.

   O isolamento temporal entre as fases TT e ET é garantida por meio de um *offset* global, que é calculado após a recepção e decodificação da TM. O processo de enfileiramento de mensagens segue a prioridade de cada uma, uma vez que reordena o *buffer* de saída de acordo com a mensagem prioritária definida pelo seu identificador. Isto impede que mensagens de alta prioridade sejam bloqueadas na fila devido à mensagens de baixa prioridade.

   Uma vez que o módulo `FTT-Station` é informado sobre a recepção da TM, inicia um temporizador interno, a fim de programar o início da fase TT seguinte no instante apropriado. O cálculo deste instante considera a quantidade de mensagens TT presentes em cada ciclo EC, e é calculado dinamicamente pelos nós a cada recepção da TM. As mensagens ET que tenham sido previamente carregadas para transmissão são enviados se, e somente se houver tempo suficiente antes do inicio da fase TT. Isto é implementado através da ativação /desativação do *buffer* de saída correspondente do controlador CAN.

   A troca de mensagens na fase TT é realizado de forma autónoma, isto é, a transmissão e a recepção são realizadas no nível do protocolo FTT-CAN sem interferência no nível de aplicação. A transmissão é realizada através da uma sequência carregada no instante da decodificação da TM. Na fase ET a troca de mensagens requer pedidos implícito da aplicação através de uma API específica. Cada requisição é enfileirada de acordo com a prioridade da mensagem.

4. Finalmente, o módulo *FTT-Master* é responsável pela escalonamento dinâmico das mensagens TT. Antes de enviar a mensagem TM, o nó mestre define a sequência de mensagens que devem ser transmitidas baseado em um método de análise de escalonabilidade que considera requisitos temporais como a período T_i e prioridade P_i, pior caso de tempo de transmissão C_i, defasagem temporal Ph_i e *deadline* D_i. Por outro lado tráfego ET é agendado com base no método por prioridade fixa. Entretanto, a estratégia de escalonamento *on-line* realizado pelo nó mestre não foi abordado neste trabalho. Assim, um nó mestre estático foi implementado apenas para disparar as mensagens TT especificadas para o estudo de caso proposto.

No processo de inicialização do kernel dos nós escravos e nó mestre seus respectivos módulos devem ser carregados, módulo `FTT-Station` e `FTT-Master`, respectivamente.

5.3 Validação do sistema

5.3.1 Arquitetura

A fim de validar o protocolo FTT-CAN sobre o μClinux com RTAI, um projeto em andamento foi utilizado como banco de teste. A idéia principal do desse projeto era criar uma arquitetura automotiva *drive-by-wire* o que poderia ser usada como plataforma eletro-eletrônica para futuros trabalhos no Departamento de Engenharia Eléctrica da UFRGS.

A arquitetura proposta inclui algumas funções veículares, tais como:

1. *Steer-by-wire*;
2. Assistência a estacionamento;
3. Indicação de velocidade veículo;
4. Indicação de nível de combustível;
5. Indicação de temperatura motor;

Com exceção da primeira função, todas as funções estão presentes no nó painel de instrumentos que é responsável por apresentar as respectivas informações para o condutor. A função de assitência de estacionamento possui uma visualização interativa no painel de instrumento, que possui parte da função distribuída entre os dois nós com sensores de proximidade (fixada na parte traseira e dianteira do veículo). Uma imagem dinâmica apresentando o comportamento dos sensores dianteiros e traseiros no painel de instrumentos é prevista.

Com essas informações sobre as funções veículares previstas, todas devem ser mapeados a fim de alcançar um modelo do sistema com todos os parâmetros de tarefas e mensagens definidas. A Figura 43 apresenta este processo.

O primeiro passo é o mapeamento de todas as funções entre ET e TT abordagens, a fim de definir quais funções serão baseadas em evento e as que serão cíclicas. Este é um processo importante onde parâmetros relacionados com as mensagens e as tarefas são definidas, por exemplo: ET ou TT, prioridade, período, fase e *deadline*.

Depois disso, com todas as mensagens e as tarefas já definidas, as funções são distribuídas entre os nós distrubuídos em pontos estratégicos no veículo. Geralmente, esse processo é iniciado com um quantidade definida de nós, devido aos custos relacionados. A seguir as funções distribuídas entre os nós são apresentadas:

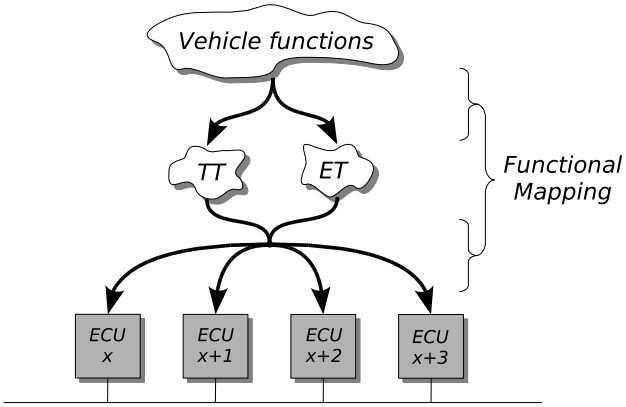

Figura 43: Mapeamento de funções de sistema

- ECU1 - leitura do estado do sistema de assistência de estacionamento dianteiro e temperatura do motor ;
- ECU2 - leitura da velocidade do veículo, ângulo das rodas e controle dos ângulo das rodas de acordo com o ângulo volante;
- ECU3 - FTT-CAN mestre, gera o mensagem TM;
- ECU4 - leitura do ângulo do volante e controle do *force-feedback*;
- ECU5 - leitura de todas as informações: velocidade veículo, temperatura do motor, nível tanque de combustível, e estado do sistema de assistência de estacionamento;
- ECU6 - leitura do estado do sistema de assistência de estacionamento traseiro e nível de tanque de combustível.

Um esquema simplificado da arquitetura proposta com suas unidades de controle eletrônico (ECU - *electronic control unit*) ou nós é representada na Figura 44.

5.3.1.1 Plataforma de Hardware

Há duas plataformas de hardware distintas, uma para ECUs, que são componentes presentes no corpo e sistema eletrônico do veículo - sistema de assistência de estacionamento e *steer-by-wire*, respectivamente. E outra plataforma de hardware para o painel de instrumentos.

Cada ECUS é composta por um hardware baseado em microprocessadores Freescale Coldfire 5282 66MHz , que tem um controlador CAN embutido, 16MB de memória RAM e 2MB de memória Flash.

O painel de instrumentos é uma plataforma de hardware com processador Freescale 5200 PowerPC. Esta placa tem 64MB de memória RAM e 32MB de memória flash, o processador executa a 400Mhz. Os principais dispositivos do processador são: CAN, PCI (Peripheral Component Interconnect) e controladores Ethernet embutido. Montado no slot PCI esta uma placa de vídeo, este placa é conectado a um LCD de 8” de largura de tela touch-screen.

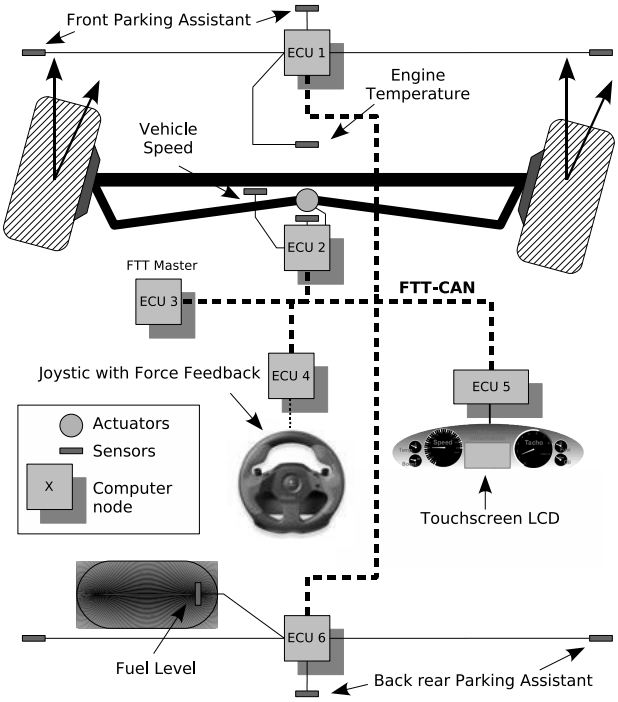

Figura 44: Arquitetura Baja-by-Wire

5.3.1.2 Painel de Instrumentos

No projeto Baja-by-wire foi proposto o uso de um conjunto de hardware e software em conjunto LCD touch-screen, em substituição de indicadores analógicos - que geralmente usam motores de passo - no painel de instrumentos do automóvel. Embora esta solução pode ter um alto custo em relação aos paineis estáticos com indicadores mecânica esta abordagem proporciona algumas melhorias, tais como:

- Sem problemas mecânicos causados por desgastes;
- *Themability* - aparência personalizavél através de *skins*;
- Configurabilidade - indicadores analógicos ou digitais podem ser escolhidos para cada função;
- Alarms - como se trata de sistema configuravél, alarmes podem ser criados para indicação de condições de erro ou avarias.

Atualmente peças mecânicas utilizadas nos painéis automotivos são sujeitos a desgastes causado por atrito. Com esta nova abordagem estes problemas podem ser eliminados considerando um custo diferenciado do componente tradicional. Themability é um recurso interessante onde usuário pode definir o "look and feel"do seu painel de instrumentos. Então, usuário pode escolher uma aparência que se identificar consigo. Em alguns casos o usuário pode personalizar o comportamento do painel de instrumentos para exibir informações de acordo com suas necessidades. Usando um LCD touch-screen o usuário pode fazer tuda customização através de toques na tela e selecionar características desejadas.

O painel de instrumentos possui uma arquitetura de software baseada no Linux com a extensão RTAI, porém todas as funções do painel de instrumentos são executadas em

modo usuário normal (na espaço de usuário). Mas o protocolo FTT-CAN deste componente foi implementado usando os recursos RTAI e se comunica com as aplicações usando IPC (*Inter Process Communication*) e partilhado principalmente memória FIFOs. Em vez desenvolver todos os componentes gráficos, este projeto propõe uma abordagem: usar um HMI que exiga pouco recurso computacional para compor todas as informações de telemetria. Em o lado esquerdo da Figura 45 é apresentada a arquitetura de software usado no painel de instrumentos. O HMI é o software Lintouch, que possui código fonte aberto e é utilizados principalmente como interface para processos de automação industriais. Lintouch tem algumas características importantes para o projeto:

Multiplataform: Lintouch foi desenvolvido em linguagem C/C++ usando bibliotecas gráficas Qt, desta forma tendo poucas dependências e possui suporte para a plataforma de hardware do painel de instrumentos (PowerPC).

Lightweight: como citado antes, é sistema que exige pouco poder computacional. O tamanho do servidor pós-compilação é de aproximadamente 100 KB e o cliente aproximadamente 900 KB.

Extensible: a arquitetura de software do Lintouch é modular, cada recurso (*plugin* ou *template*) pode ser adicionado apenas copiando uma biblioteca.

Opensource: Lintouch é software *opensource*. Todas código fonte são liberados sob a licença GNU GPL.

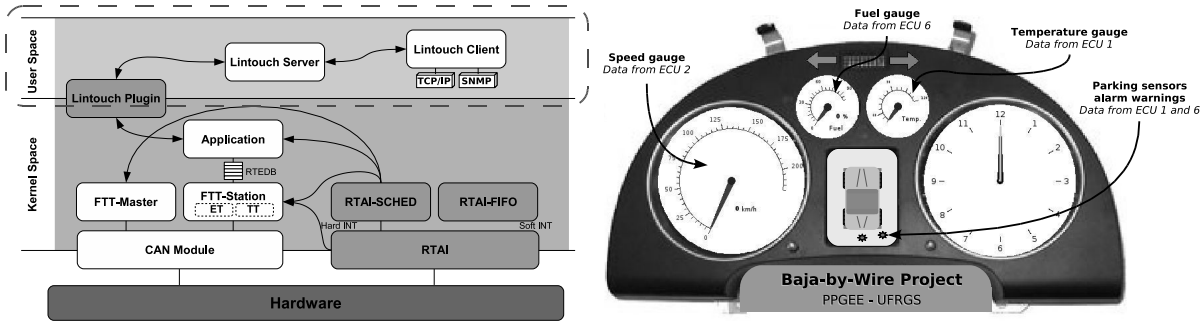

Figura 45: Arquitetura de software e imagem do quadro de instrumentos

Para utilizar Lintouch no projeto é necessário criar um *plugin* para FTT-CAN, para obter todas as informações de telemetria de outros nós da rede. No lado direito da Figura 45 é apresentado uma imagem do protótipo funcional com seus componentes.

Na webpage[1] do projeto além da disponibilidade de todos os códigos FTT-CAN também está disponível um vídeo de demonstração do protótipo funcional do painel de instrumentos em execução. A conclusão dessa dissertação foi obtida antes da conclusão do projeto "Baja-by-wire".

5.3.2 Resultados práticos

O conjunto de mensagens e tarefas TT da aplicação são apresentadas nas Tabelas 2 e 3, respectivamente.

[1]http://sourceforge.net/projects/fttcan4rtai/ - webpage do projeto 'FTT-CAN for RTAI' apresentado nesta dissertação.

Tabela 2: Conjunto de mensagens TT do sistema Baja-by-Wire

Msg	Description	ECU	P_i	T_i	Ph_i
1	Ângulo para atuação	4	7	5	0
2	Velocidade veículo	2	6	5	2,5
3	Ângulo das rodas	2	5	5	2,5
4	Temperatua motor	1	4	500	5
5	Nível combustível	6	3	500	10
6	Sensor colisão frontal	1	2	200	15
7	Sensor colisão traseiro	6	1	200	20

A prioridade de mensagem, P_i, na Tabela 2 e a identificação de tarefa, Id_i, na Tabela 3 são diretamente mapeados para *flags* de mensagem e tarefa na mensagem TM enviada pelo nó FTT-CAN mestre. A oitavo coluna da Tabela 3 apresenta as mensagens consumidas (C) e produzidas (P) pelas respectivas tarefas TT da aplicação. O ciclo elementar (EC) do FTT-CAN para esta aplicação é de 2,5 ms para garantir a integração progressiva das mensagens e tarefas de mesmo período. O menor período de mensagem e de tarefa é de 5ms. Todas as medidas foram feitas por um conjunto de 3000 amostrados de mensagens e tarefas, com uma CAN de 250kbps e 4μs de resolução de *timer*.

Tabela 3: Conjunto de tarefas TT do sistema the Baja-by-Wire

Id_i	Descrição	ECU	T_i	Ph_i	C/P Msg
0	Amostragem do ângulo do volante	4	5	0	P Msg1
1	Controle do ângulo do volante	2	5	2,5	C Msg1
					P Msg3
2	Amostragem da velocidade do veículo	1	5	2,5	P Msg2
3	Controle *feedback* do volante	4	5	5	C Msg2
					C Msg3
4	Amostragem temperatura motor	1	500	5	P Msg4
5	Amostragem nível de combustível	6	500	10	P Msg5
6	Amostragem sensor de colisão frontal	1	200	15	P Msg6
7	Amostragem sensor de colisão traseiro	6	200	20	P Msg7
8	Indicação temperatura motor	5	500	7,5	C Msg4
9	Indicação nível de combustível	5	500	12,5	C Msg5
10	Indicação sensor colisão frontal	5	200	17,5	C Msg6
11	Indicação sensor colisão traseiro	5	200	22,5	C Msg7

5.3.2.1 Tráfego de mensagens TT

O *jitter* mensurado é apresentado para a mensagem TM e mensagens 1, 2, 5, 7. De acordo com os parâmetros da Tabela 2, as mensagens 3, 5 e 7 são semper as primeiros da fase TT do EC devido suas prioridades P_i e defasagem Ph_i, e por isso estas sofrem maior impacto do efeito bloqueio. As mensagens 1 e 2 estão no fim da fase TT, no entanto, estas sofrerem pessimismo do método *bit stuffing*. As medidas sem a abordagem com *offset* (secção 4.4.1) atingiu um alto grau de bloqueio, 11,6% e 10,2% para as mensagens 5 e 7, respectivamente. Além disso, um release *jitter* entre 28μ(J_{max}) e -28μ(J_{min} para as mensagens 1 e 2 geradas pelo efeito *bit stuffing*. A Tabela 4 apresenta os resultados da abordagem proposta. J_{max} e J_{mim} são as máximas e mínimas variações (*jitter*), σ é o

desvio-padrão e $\%$ é a porcentagem de ocorrências do efeito bloqueio.

Tabela 4: Resultados com método offset

Message	J_{max}	J_{min}	σ	% Blocking
1	16	-16	3,58	0
2	16	-16	3,53	0
5	4	-4	2,80	0
7	8	-4	2,53	0
TM	4	-4	2,02	0

A adoção da abordagem proposta leva a uma melhoria do *jitter* imposto pelos *stuff* bits e elimina completamente as situações de bloqueio.

5.3.2.2 Execução de tarefas TT

O Tabela 5 apresenta o *jitter* de liberação da tarefa produtora $Id0$, e a tarefa consumidora /produtora $Id1$ usando abordagem com janelas de execução (apresentada na secção 4.4.3).

Tabela 5: Resultados com método slot de tarefas

Task (Id_i)	J_{max}	J_{min}	σ
0	12	-8	3,56
1	12	-8	3,59

Os resultados foram similares para outras tarefas. Figura 46 apresenta uma imagem de um ciclo EC em um osciloscópio. O canal 1 mostra a execução do processo do protocolo FTT-CAN, isto é, a processo decodificador da mensagem TM, o processo de liberação de mensagem TT e a execução de uma tarefa TT produtora dentro da janela. O canal 2 mostra o sinal do barramento CAN com o ciclo EC, contendo a mensagem TM, as fases ET e TT.

A proposta permite algumas vantagens em relação a abordagem original do protocolo FTT-CAN, que são: introdução da capacidade de um sistema ser integrado junto a outros sem perda e danos nos requisitos temporais de aplicação (*composability*), sendo possível com os instantes fixos de transmissão e execução de mensagens e tarefas; utilização de mensagens TT com comprimento de dados diferentes sem degradação do tempo de resposta de outras mensagens do sistema; garantia de tempo de resposta de mensagens TT mesmo em condições de falhas de outros nós, e a restrição de execução de tarefas TT dentro de sua respectiva fase (ou próximo) reduzindo o impacto no tempo de resposta de eventos.

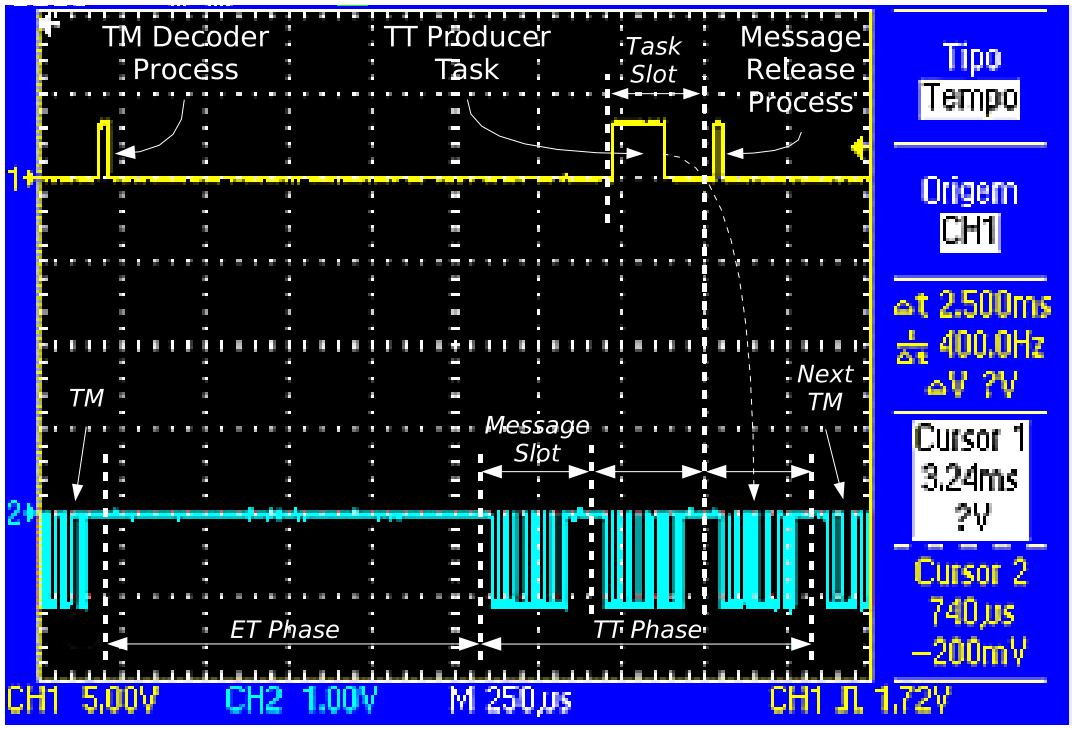

Figura 46: Resultado da implementação, ciclo EC com execução de uma tarefa TT

6 CONCLUSÃO

Este trabalho apresenta uma discussão sobre duas abordagens que estão presentes em todo e qualquer projeto de sistema distribuído de tempo real (SDTR), *event-* e *time-triggered*. Estas abordagens ditam o comportamento da execução de tarefas e transmissão de mensagens. Com estas abordagens, foi discutido escalonamento de tarefas e mensagens com as políticas de prioridade fixa e cíclico estático, no ponto de vista de sistemas de controle via rede.

É notável que a coexistência de ambas abordagens, TT e ET, a fim de garantir tanto o atraso de controle (usando a política SC) quanto o tempo de resposta a eventos externos (usando política PF) é a melhor solução para projetos SDTR. Esta coexistência pode oferecer maior flexibilidade do que aquela que privilegia uma delas. O tempo global de sistema pode reduzir tanto o bloqueio quanto as interferências na condição de escalonamento seguindo a política por PF garantindo que o *offset* seja atribuído apropriadamente. Diferentes métodos foram apresentados no capítulo 4.1, cada um com foco diferente, quer seja flexibilidade ou menos *overhead*.

O protocolo FTT-CAN, que é o foco desta dissertação, foi discutido, onde os subsistemas FTT síncrono e assíncrono foram apresentados.

Durante este trabalho foram identificadas alguns problemas que impactam no tempo de resposta de tarefas e mensagens. O primeiro deles é quando há uma execução de tarefa precedendo a transmissão de uma mensagem, onde qualquer variação em função de da execução ou da liberação da tarefa induz atrasos na transmissão da respectiva mensagem. Esse problema podem gerar defasagem temporal na liberação dessas tarefas, que, por consequência gera variação no tempo de resposta da mesma e, assim, pode gerar condições de bloqueio do ponto inicial da fase síncrona. Outro aspecto negligenciado no protocolo original, em relação à transmissão de mensagens, foi a variabilidade do tempo de transmissão gerado pelo método *bit-stuffing* nativo do protocolo CAN que aumentam o comprimento das mensagens gerando atrasos. Estes atrasos se somam quando, em uma mesma fase síncrona de um dado ciclo EC, existem várias mensagens. Assim,a última mensagem terá o maior impacto, ou seja, sempre terá o maior *jitter*.

Outro aspecto identificado é quanto ao escalonamento de tarefa, onde não existia uma janela de execução especifica para tarefas síncronas. Assim, uma tarefa síncrona era liberada logo em seguida da codificação da mensagem TM no inicio do ciclo EC e acima da fase assíncrona. Este especto pode gerar interferências nas tarefas e mensagens assíncronas e consequentemente prejudicar seus respectivos tempos de resposta.

No ponto de vista de sistema de controle via rede, as consequências destes aspectos identificados podem ser mais severas, porque pode afetar a dinâmica do processo pelo aumento ou grande variações do atraso de controle.

REFERÊNCIAS

ALBERT, A. Comparison of event-triggered and time-triggered concepts with regard to Distributed Control Systems. In: EMBEDDED WORLD, 2004, Nurnberg, Germany. **Proceedings …** [S.l.]: IEEE, 2004. p.235-252.

ALMEIDA, L. A word for operational flexibility in distributed safety-critical systems. In: IEEE INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 8., 2003, Guadalajara, Mexico. **Proceedings …** [S.l.]: IEEE, 2003. p.233.

ALMEIDA, L.; FONSECA, J. A. Analysis of a simple model for non-preemptive blocking-free scheduling. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 13., 2001. Washington, DC, USA, **Proceedings …** [S.l.]: IEEE, 2001. p.233.

ALMEIDA, L.; PASADAS, R.; FONSECA, J. Using the planning scheduler to improve flexibility in real-time fieldbus network. **IFAC - Iternational Federation of Automatic Control**, [S.l.], v.7, p.101–108, 2001.

ALMEIDA, L.; PEDREIRAS, P.; FONSECA, J. The FTT-CAN protocol - why and how. **IEEE Transactions on Industrial Electronics**, [S.l.], v.49, n.6, p.1189– 1201, Dec. 2002.

ANDERSSON, M. P.; LINDSKOV, J.-H. **Real-time linux in an embedded environment:** a port and evaluation of RTAI on the CRIS architecture. 2003. 200 f. Dissertação (Mestrado em Engenharia Elétrica) - Lund Institute of Technology, Sweden, 2003.

CALHA, M.; FONSECA, J. Adapting FTT-CAN for the joint dispatching of tasks and messages. In: IEEE INTERNATIONALWORKSHOP FACTORY COMMUNICATION SYSTEMS, 4., 2002, Västerås, Sweden. **Proceedings …** [S.l.]: IEEE, 2002.

CALHA, M.; SILVA, V.; FONSECA, J. Kernel design for FTT-CAN systems. In: IEEE INTERNATIONAL WORKSHOP ON FACTORY COMMUNICATION SYSTEMS, 6., 2006, Torino, Italy. **Proceedings …** [S.l.]: IEEE, 2006.

CAN in Automation (CiA). Disponível em: <http://www.can-cia.org/>. Acesso em: 25 nov. 2010.

CAN physical layer. Disponível em: <http://www.can-cia.org/can/physical-layer/>. Acesso em: 23 nov. 2010.

CONSORTIUM, F. **FlexRay communications system protocol specification version 2.0.** [S.l.]: FLEXRAY CONSORTIUM, 2004c.

CONSORTIUM, L. **LIN specification package revision 2.1**. [S.l.: s.n.], 2006.

DIONNE, J.; DURRANT, **M. Embedded Linux Microcontroller Project.** Disponível em: <http://www.uclinux.org>. Acesso em: 26 nov. 2010.

EBNER, C. Efficiency evaluation of a time-triggered architecture for vehicle body-electronics. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 10., 1998, Berlin, Germany. **Proceedings …** [S.l.: s.n.], 1998.

GRZEMBA, P. D. I. A. **MOST:** the automotive multimedia network. [S.l.]: Franzis, 2007.

KOPETZ, H. **Real-time systems:** design principles for distributed embedded applications. l. ed. Norwell: Kluwer Academic Publishers, 1997.

KOPETZ, H.; BAUER, G. The time-triggered architecture. **Proceedings of the IEEE,** [S.l.], v.91, n.1, p.112–126, 2001.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. **Journal of the ACM**, [S.l.], v.20, n.1, p.46–61, 1973.

LONN, H.; AXELSSON, J. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In: EUROMICRO CONFERENCE ON REAL TIME SYSTEMS, 11., 1999, York, England. **Proceedings ...** [S.l.: s.n.], 1999. p.142-149.

RTAI project. Disponível em: <http://www.rtai.org/>. Acesso em: 23 nov. 2010.

MARTINS, E.; ALMEIDA, L.; FONSECA, J. A. An FPGA-based coprocessor for realtime fieldbus traffic scheduling: architecture and implementation. **Journal of Systems Architecture:** the EUROMICRO journal, New York, NY, USA, v.51, n.1, p.29–44, 2005.

NOLTE, T.; HANSSON, H.; NORSTRÖM, C. Effects of varying phasings of message queuings in CAN based systems. In: INTERNATIONAL CONFERENCE ON REAL-TIME COMPUTING SYSTEMS AND APPLICATIONS (RTCSA'02), 8., 2002. Tokyo, Japan. **Proceedings …** [S.l.]: IEEE, 2002. p.261–266.

NOLTE, T.; HANSSON, H.; NORSTRÖM, C. Probabilistic worst-case response-time analysis for the controller area network. In: IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM (RTAS'03), 9., 2003, Washington, DC, USA. **Proceedings …** [S.l.]: IEEE, 2003. p.200–207.

POP, T. **Scheduling and optimisation of heterogeneous time and event-triggered distributed embedded systems**. 2003. 113 f. Tese (Doutorado em Engenharia Elétrica) - Escola de Engenharia de Linköpings, Linköpings, 2003.

RAHUL SHAH, X. D. **An introduction to TTCAN (time triggered tontroller area network)**. Disponível em: <http://cs.uni-salzburg.at/ck/teaching>. Acesso em: 19 nov. 2010.

SUN, J.; LIU, J. Synchronization Protocols in Distributed Real-Time Systems. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 16., 1996, Hong Kong. **Proceedings …** [S.l.: s.n.], 1996. p.38–45.

SUNDARAMOORTHY, N. **Put a configurable 32-bits processor in FPGA.** Disponível em: <http://www.embedded.com/>. Acesso em: 19 nov. 2010.

TINDELL, K.; BURNS, A.; WELLINGS, A. Calculating controller area network message response times. **Control Engineering Practice**, [S.l.], v. 3, n. 8, p.123-127, 1995.

TINDELL, K.; CLARK, J. Holistic schedulability analysis for distributed hard real-time systems. **Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)**, [S.l.], v.40, p.117–134, 1994.

TTA-GROUP. **Time-triggered protocol TTP/C high-level specification document protocol version 1.1**. [S.l.]: TTA-Group, 2003.

USA, M.-B. **Domestic Digital Bus (D2B)**. Disponível em: <http://www.mercedestechstore.com/>. Acesso em: 23 nov. 2010

YODAIKEN, V.; BARABANOV, M. **A real-time linux**. Disponível em: <http://rtlinux.cs.nmt.edu/rtlinx/u.pdf>. Acesso em: 23 nov. 2010

7 APÊNDICE A

7.1 Publicações realizadas durante o período

Seguindo o escopo deste trabalho, as seguintes publicações foram realizadas:

1. Fernando H. Ataide, Carlos E. Pereira, Valter F. Silva, *A New Approach for Time-Triggered Phase in the FTT-CAN Protocol - A Case Study in an Automotive System*: 27th IEEE Real-Time Systems Symposium - **RTSS'2006**, Rio de Janeiro, December, 2006.

2. Fernando H. Ataide, Alan C. Assis, Carlos E. Pereira, *Automotive X-by-Wire Systems based on Linux – An Open Source Project*: Eighth Real-Time Linux Workshop, **RTLWS'2006**, China, October 12-15, 2006.

3. Fernando H. Ataide, Carlos E. Pereira, Walter F. Lages, Alan C. Assis, *On the design of an embedded FTT-CAN platform with improvement of its inherent jitter*: 4th International IEEE Conference on Industrial Informatics **INDIN'2006**, Singapore, 16-18 Agosto, 2006.

4. Fernando H. Ataide, Carlos E. Pereira, Alan C. Assis, *An embedded communication platform with an enhanced FTT-CAN*: 8th Brazilian Workshop on Real-Time Systems - **WTR'2006**, Curitiba, Brasil, Junho, 2006.

5. Fernando H. Ataide, Fabiano Carvalho, Carlos E. Pereira, *A Comparative Study of Embedded Protocol For Safety Critical Control Application*: 12th IFAC Symposium on Information Control Problems in Manufacturing - **INCOM'2006**, Saint-Etienne - França, Maio, 2006.

6. Fernando H. Ataide, Carlos E. Pereira, Alan C. Assis, *An embedded communication platform based on Linux for automotive systems*: Forum Internacional de Software Livre - **FISL7**, april 2006.

7. Fernando H. Ataide, Carlos E. Pereira, Fabiano C. Carvalho, *Analysis of Communication Requirements in Automotive Buses*: In Proceedings of the Society of Automotive Engineers International Conference, **SAE2005**, novembro 2005.

8. Fernando H. Ataide, Carlos E. Pereira, Edison P. Freitas, Elias T. Silva, *Performance evaluation of Java Architectures in Embedded Real-Time Systems*: In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, **ETFA'2005**, setembro 2005.

8 APÊNDICE B

8.1 Código fonte do FTT-CAN

O código fonte deste trabalho esta disponível na página do projeto no site SourceForge.

Nome do projeto no SourceForge: **fttcan4rtai**

Link: **http://sourceforge.net/projects/fttcan4rtai/**