RODRIGO LANGE

Estudo Sobre a Análise de Escalonabilidade de Barramentos do Tipo FlexRay

FLORIANÓPOLIS
2010

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Estudo Sobre a Análise de Escalonabilidade de Barramentos do Tipo FlexRay

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia
de Automação e Sistemas.

Rodrigo Lange

Florianópolis, Agosto de 2010.

Estudo Sobre a Análise de Escalonabilidade de Barramentos do Tipo FlexRay

Rodrigo Lange

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Automação e Sistemas, Área de Concentração em *Controle, Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.'

Rômulo Silva de Oliveira, Dr.
Orientador

Nestor Roqueiro, Dr.
Co-orientador

Jose Eduardo Ribeiro Cury, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Rômulo Silva de Oliveira, Dr.
Orientador

Gerson Battisti, Dr.

Luis Fernando Friedrich, Dr.

Marcelo Ricardo Stemmer, Dr.

AGRADECIMENTOS

Inicialmente, quero agradecer ao meu orientador Rômulo Silva de Oliveira pelo incentivo, paciência, dedicação e por acreditar sempre que o trabalho poderia dar frutos. Agradeço da mesma forma ao meu coorientador Nestor Roqueiro que de uma maneira especial mesmo à distância acompanhou o trabalho.

Agradeço à minha família pela confiança em mim depositada e por saber que posso contar sempre com ela. Agradeço especialmente à minha mãe por proporcionar à mim e à minha irmã uma educação especial e íntegra.

Agradeço também aos amigos do LTIC pela amizade, compartilhamento de conhecimento, trilhas e festas.

Por fim, agradeço ao CNPQ pelo suporte financeiro que tornou possível este trabalho.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

Estudo Sobre a Análise de Escalonabilidade de Barramentos do Tipo FlexRay

Rodrigo Lange

Agosto/2010

Orientador: Rômulo Silva de Oliveira, Dr.
Co-orientador: Nestor Roqueiro, Dr.
Área de Concentração: Controle, Automação e Sistemas
Linha de Pesquisa: Sistemas Computacionais
Palavras-chave: FlexRay, Redes Intra-Veiculares, Redes de Comunicação Automotivas
Número de Páginas: xxiv + 133

O Sistema de Comunicação FlexRay tem sido promovido na literatura como o futuro padrão de fato em comunicações automotivas que requerem desempenho e segurança, como em futuros sistemas *X-by-Wire*. O objetivo primário deste trabalho é avaliar experimentalmente as técnicas da literatura que abordam a análise do tempo de resposta de mensagens alocadas no segmento dinâmico do FlexRay, visando criar uma documentação para auxiliar o projetista de sistemas veiculares na escolha da técnica adequada para a análise de cenários diferentes. Devido a limitações encontradas nos métodos existentes, são apresentados quatro novos métodos para a análise, sendo que três dos métodos são heurísticas para uso com sistemas onde mensagens esporádicas são alocadas no segmento dinâmico do FlexRay.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.

On the Schedulability Analysis of the FlexRay Network Protocol

Rodrigo Lange

August/2010

Advisor: Rômulo Silva de Oliveira, Dr.
Co-advisor: Nestor Roqueiro, Dr.
Area of Concentration: Control, Automation and Systems
Research Area: Computation Systems
Key words: FlexRay, In-Vehicle Networks, Network Communication Protocols
Number of Pages: xxiv + 133

The FlexRay Communication System has been promoted in the literature as the future de facto standard for automotive communications in safe-critical applications, such as future X-by-Wire systems. The primary objective of this work is to experimentally evaluate techniques from the literature addressing the analysis of the response time of messages allocated to the dynamic segment of FlexRay, in order to create documentation that aids the designer of vehicular systems. Due to limitations found in existing methods, this work presents four new methods for the analysis. Three methods are heuristics for use with systems in which sporadic messages are allocated to the dynamic segment of FlexRay.

Conteúdo

Lista de Siglas e Termos

C	Tempo de Computação de uma mensagem ou tarefa
CAN	*Controller Area Network*
Cluster	Sistema de comunicação FlexRay com múltiplos nós conectados por no mínimo um canal de comunicação
CHI	*Controller Host Interface*
CPU	*Central Processing Unit*
CSP	Processo de Sincronização de Relógios (*Clock Synchronization Process*)
CTL	*Computation Tree Logic*
D	*Deadline* relativo de uma mensagem ou tarefa
DYN	Segmento Dinâmico (*Dynamic Segment*)
ECU	*Electronic Control Unit*
ET	*Event-triggered*
ST	Segmento Estático (*Static Segment*)
DYN_{bus}	Tamanho do Segmento Dinâmico
ST_{bus}	Tamanho do Segmento Estático
FC	*FlexRay Cycle*
FIFO	*First In, First Out*
FPS	*Fixed Priority Scheduling*
FTDMA	*Flexible TDMA*
FTM	*Fault-Tolerant Midpoint Algorithm*
GPC	*Greedy Processing Component*
ILP	*Integer Linear Programming*

LIN	*Local Interconnect Network*
LTL	*Linear Temporal Logic*
MAC	*Media Access Control*
MIT	*Minimum Interarrival Time*
MMC	Mínimo Múltiplo Comum
MOST	*Media Oriented System Transport*
MS	*Minislot*
MT	*Macrotick*
μT	*Microtick*
MTG	Processo de Geração de Macroticks (*Macrotick Generation Process*)
NIT	*Network Idle Time*
P	Período de uma mensagem ou tarefa
PN	*Petri Net*
R	Tempo de Resposta de uma mensagem ou tarefa
RM	*Rate Monotonic*
RTC	*Real-Time Calculus*
SAE	*Society for Automotive Engineers*
SCS	*Static Cyclic Scheduling*
TDMA	*Time Division Multiple Access*
TPN	*Time Petri Net*
TT	*Time-triggered*
WCRT	*Worst-Case Response Time*

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Capítulo 1

Introdução

A indústria automotiva é atualmente um dos maiores setores econômicos mundiais, produzindo anualmente dezenas de milhões de veículos e contribuindo significativamente para a receita de vários países. E nos últimos anos esta indústria tem observado um crescimento exponencial no número das funções computadorizadas embarcadas em veículos. Toda uma classe de funções eletrônicas para sistemas de navegação, controle de tração, controle de estabilidade, sistemas de segurança ativa, multimídia e muitos outros tem sido desenvolvidos, sendo que veículos modernos possuem mais de 70 controladores eletrônicos trocando 2.500 sinais que representam informações elementares como velocidade ou temperatura do motor (NAVET; SIMONOT-LION, 2008).

Este crescimento na complexidade das arquiteturas eletrônicas, aliado à restrições relacionadas a sensores e atuadores levou à adoção de uma abordagem distribuída para a implementação das várias funções existentes. Neste contexto, as redes e protocolos de comunicação tem importância fundamental pois servem como suporte para a integração de funções, reduzindo o custo e a complexidade do cabeamento e fornecendo também um meio para a implementação de métodos de tolerância a falhas.

Tipicamente, o sistema embarcado em um veículo é dividido em

domínios funcionais que correspondem a diferentes características e restrições. Dois destes domínios estão relacionados especificamente ao controle em tempo real do comportamento do veículo, sendo o "*power train*" relacionado ao controle do motor e transmissão e o "*chassis*" relacionado ao controle da suspensão, direção e freios. Nestes domínios, aspectos relacionados à segurança são fundamentais e as soluções técnicas devem garantir a confiabilidade do sistema, mantendo-se ao mesmo tempo rentáveis (NAVET; SIMONOT-LION, 2008). Para estes domínios, o protocolo de comunicação *Controller Area Network* (CAN) tem sido o padrão de fato, com mais de 400 milhões de dispositivos com *interface* para este protocolo fabricados a cada ano (DAVIS et al., 2007).

Existe atualmente na indústria automotiva a tendência de se substituir sistemas mecânicos ou hidráulicos por outros eletrônicos, existindo duas razões para esta mudança, uma econômica e outra tecnológica. A razão econômica está relacionada ao fato de que o custo de componentes de *hardware* vem decaindo, enquanto a confiabilidade dos mesmos aumenta. A razão tecnológica é que os sistemas eletrônicos permitiram a adoção de novas funções cujo desenvolvimento seria impossível se utilizados apenas sistemas mecânicos ou hidráulicos. Esta tendência resultou no conceito *X-by-Wire*, onde sistemas eletrônicos ou hidráulicos presentes em veículos são substituídos por outros puramente eletrônicos. Historicamente, a primeira aplicação crítica para o *X-by-Wire* foi um sistema *Throttle-by-wire* implementado em um Chevrolet Corvette em 1980, onde o tradicional sistema de acelerador por cabo foi substituído por outro eletrônico. Sistemas eletrônicos para o câmbio (também chamados *Shift-by-Wire* ou *Gear-by-Wire*) já estão presentes em alguns veículos de luxo como os das séries 5 e 7 da BMW (ZURAWSKI, 2005).

Segundo vários autores, o CAN não é apropriado para futuras aplicações *X-by-Wire* por possuir sérias limitações em relação a aspectos como desempenho e segurança, e por isso foram desenvolvidos novos protocolos para aplicações automotivas, como por exemplo o *Byteflight* da BMW (ZURAWSKI, 2005; SCHEDL, 2007). Mas o protocolo apon-

tado como futuro padrão de fato para uso em aplicações automobilísticas é aquele chamado **Sistema de Comunicação FlexRay** (NAVET; SIMONOT-LION, 2008; POP et al., 2008; ZURAWSKI, 2005).

1.1 Conceitos Básicos e Motivação

O Sistema de Comunicação FlexRay é um barramento serial digital para uso em aplicações automotivas que visa atender às demandas de largura de banda, confiabilidade, determinismo e sincronização em futuros sistemas automotivos como os relacionados a *X-by-Wire*. Entre outros aspectos, fornece flexibilidade, largura de banda e determinismo através da combinação de métodos estáticos e dinâmicos para a transmissão de mensagens, incorporando as vantagens de protocolos síncronos e assíncronos bem conhecidos. O FlexRay foi desenvolvido entre os anos 2000 e 2009 por uma aliança de fabricantes que incluiu entre outros BMW, DaimlerChrysler e Bosch (FLEXRAY, 2005).

Historicamente, são identificados cinco domínios funcionais em veículos: *power train* (relacionado ao motor e transmissão), *chassis* (rodas, direção, freio, posição e movimentação entre outros), *body* (relacionado a itens que não estão diretamente relacionados à dinâmica do veículo mas que fazem parte do suporte aos passageiros, como por exemplo *airbags*, luzes, ar condicionado, janelas elétricas e outros), *HMI* (engloba os sistemas de troca de informação entre o veículo e o motorista, como botões e painéis) e *telemática* (sistemas que permitem a troca de informações com o mundo exterior, como rádio, GPS e acesso à Internet) (NAVET; SIMONOT-LION, 2008).

O constante aumento nas necessidades de largura de banda, desempenho e baixo custo levou a uma diversificação dos protocolos de rede utilizados nos domínios funcionais presentes em veículos. Em 1994 a *Society for Automotive Engineers* (SAE) definiu uma classificação para protocolos de comunicação automotivos com base na velocidade de transmissão e funções que são distribuídas na rede. As redes da *Classe A* possuem uma taxa de transmissão menor que 10kb/s e são utilizadas

para transmitir dados de controle simples empregando baixa tecnologia, sendo principalmente integradas ao domínio *body*. Um exemplo de protocolo Classe A é o *Local Interconnect Network* (LIN). As redes da *Classe B* operam com velocidades entre 10 e 125 kb/s e são dedicas à transmissão de dados entre *Electronic Control Unit* (ECUs) com o objetivo de reduzir a quantidade de sensores através do compartilhamento de informações, sendo exemplo de protocolos para a Classe B o J1850 e o CAN de baixa velocidade. As redes da Classe C possuem taxas de transmissão entre 125 kb/s e 1Mb/s e são utilizadas atualmente nos domínios do *chassis* e *power train*, sendo o CAN de alta velocidade o atual padrão de fato. As redes da *Classe D* são redes de alta velocidade voltadas a sistemas multimídia sendo o *Media Oriented System Transport* (MOST) o principal protocolo atualmente utilizado para este tipo de sistema (NAVET et al., 2005).

Devido às suas características, o FlexRay tem sido citado como o futuro padrão de fato para comunicação de dados nos domínios de *chassis*, *power train* e sistemas *X-by-Wire*. Entretanto, como as necessidades específicas de outros domínios continuarão exigindo diferentes tipos de protocolos como LIN, CAN ou MOST, existe a visão de que as diversas ECUs presentes nestes domínios trocarão dados através de *gateways* interligados por um *backbone* FlexRay, como ilustrado na Figura 1.1 (SCHEDL, 2007; LORENZ, 2008; SEO et al., 2008).

Para que um sistema veicular como o ilustrado na Figura 1.1 seja projetado corretamente, as restrições temporais associadas a cada subsistema devem ser avaliadas, e esta avaliação passa pela análise do tempo de resposta no pior caso das mensagens de dados trocadas entre as diversas ECUs. Para a análise de protocolos como o CAN existem métodos bem aceitos na literatura, como por exemplo o apresentado em (DAVIS et al., 2007). Mas, como o FlexRay é um protocolo relativamente novo, não existe na literatura um método para análise de tempo de resposta de mensagens alocadas no segmento dinâmico que seja amplamente aceito, apesar da existência de propostas para esta análise.

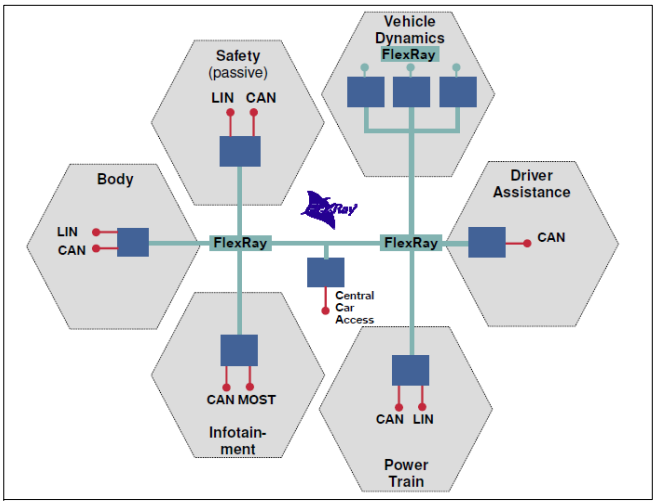

Figura 1.1: FlexRay como Backbone (SCHEDL, 2007)

1.2 Objetivos

O objetivo inicial deste trabalho é avaliar experimentalmente as técnicas da literatura que abordam a análise do tempo de resposta de mensagens alocadas no segmento dinâmico de barramentos do tipo FlexRay. Esta análise visa comparar os diversos métodos de forma a auxiliar o projetista de sistemas veiculares na escolha da técnica adequada para a análise de cenários diferentes.

Mas, como será discutido no Capítulo 3, as técnicas atualmente disponíveis apresentam falhas que levam ou a tempos de resposta otimistas para determinados sistemas, ou acrescentam demasiado pessimismo ao tempo de resposta, de modo que não são confiáveis. Por isso são apresentadas quatro novas propostas de métodos para a análise do tempo de reposta.

Estas técnicas são comparadas com os métodos existentes na literatura para se determinar qual apresenta os melhores resultados para diversos tipos de sistemas que empregam o FlexRay.

1.3 Organização do Texto

Este documento está estruturado em seis capítulos, começando com o presente onde são apresentados alguns conceitos básicos, a motivação e os objetivos do trabalho.

O capítulo 2 apresenta o Sistema de Comunicação FlexRay, descrevendo seu funcionamento e conceitos relacionados à sua especificação.

A descrição dos métodos para análise do tempo de resposta de mensagens alocadas em barramentos do tipo FlexRay existentes na literatura é feita no capítulo 3. Neste capítulo é dada ênfase à descrição dos trabalhos relacionados a análises no segmento dinâmico do FlexRay, sendo feitas considerações sobre cada proposta e apresentados resultados numéricos obtidos com a implementação das mesmas.

No capítulo 4 são apresentadas propostas de novos métodos para a análise do tempo de resposta de mensagens alocadas no segmento dinâmico do FlexRay, sendo que os resultados das avaliações experimentais destas propostas são apresentados no capítulo 5.

Por fim as conclusões do trabalho e sugestões para trabalhos futuros são apresentadas no capítulo 6.

Capítulo 2

Sistema de Comunicação FlexRay

O Sistema de Comunicação FlexRay é um barramento serial digital para uso em aplicações automotivas. Visa atender às demandas de largura de banda, confiabilidade, determinismo e sincronização em futuros sistemas automotivos como por exemplo os relacionados a *X-by-Wire*. Entre outros aspectos, fornece flexibilidade através da combinação de métodos estáticos e dinâmicos para a transmissão de mensagens, incorporando as vantagens de protocolos síncronos e assíncronos bem conhecidos.

O FlexRay foi desenvolvido entre os anos 2000 e 2009 por uma aliança de fabricantes que incluiu entre outros BMW, DaimlerChrysler e Bosch. Este consórcio concluiu seus trabalhos no ano de 2009 com a finalização da Versão 3.0 da Especificação do Sistema de Comunicação FlexRay. A especificação do sistema inclui, entre outros, definições para o Protocolo de Comunicação (*Protocol Specification*), Camada Física (*Electrical Physical Layer Specification*), Camada de Enlace de Dados (*FlexRay Data Link Layer Conformance Test Specification*) e *Bus Guardian* (*Node-Local and Central Bus Guardian Specification*) (FLEXRAY, 2005).

Neste capítulo serão descritas as principais características relacionadas à especificação do Protocolo de Comunicação. Apesar da finalização da Versão 3.0 da especificação, na data de conclusão deste trabalho apenas a versão V2.1 Rev.A (FLEXRAY, 2005) estava disponível para acesso no *site* oficial, de forma que este capítulo é baseado na mesma.

Com a finalidade de manter a clareza e coerência do texto deste capítulo em relação à especificação do FlexRay, muitos termos foram mantidos na sua forma original em inglês, como por exemplo no caso da expressão "*cluster*".

2.1 Topologias de Rede

O FlexRay é um protocolo com dois canais de comunicação que suporta taxas de transmissão de até 10 Mbit/s (FLEXRAY, 2005; SCHEDL, 2007), sendo que existem várias maneiras de projetar um *cluster*. Um *cluster* FlexRay consiste de, no máximo, dois canais de comunicação, denominados Canal A e Canal B. Cada nó da rede pode estar conectado a um ou a ambos os canais. Em uma situação normal (livre de falhas), todos os nós conectados ao Canal A são capazes de se comunicar com todos os outros nós no Canal A e, da mesma forma, todos os nós ligados ao Canal B são capazes de se comunicar com todos os outros nós ligados ao Canal B.

Os dois canais de comunicação podem ser utilizados para aumentar a largura de banda e/ou introduzir um canal redundante para aumentar o nível de tolerância a falhas (CONSORTIUM, 2005).

Um *cluster* FlexRay pode ser configurado como um barramento com um ou dois canais, em uma topologia em estrela com um ou dois canais, ou em diversas combinações híbridas de topologias tipo barramento ou estrela (FLEXRAY, 2005). Na Figura 2.1 são ilustrados exemplos de possíveis combinações para a topologia do *cluster*.

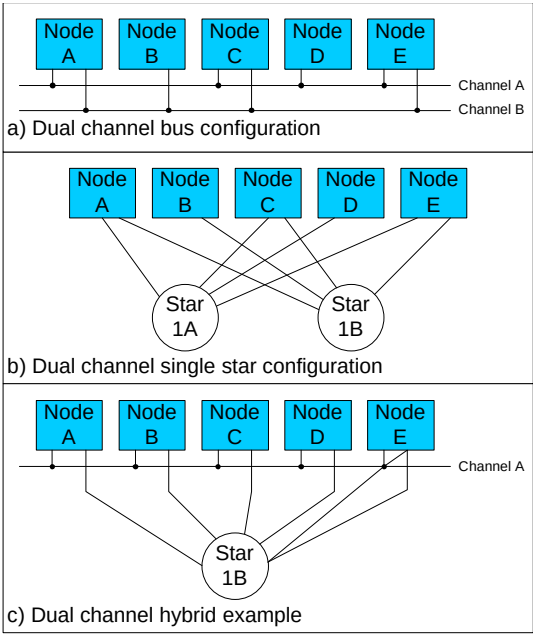

Figura 2.1: Exemplos de topologias de rede (FLEXRAY, 2005)

2.2 Controle de Acesso ao Meio

No FlexRay, o controle de acesso ao meio (*Media Access Control*, MAC) é baseado em um ciclo de comunicação com tamanho pré-definido executado periodicamente desde o início até o desligamento da rede. Cada ciclo contém um **segmento estático (*static segment*, ST)**, um **segmento dinâmico (*dynamic segment*, DYN)** e dois segmentos de protocolo, chamados ***Network Idle Time* (NIT)** e ***Symbol Window***. Na Figura 2.2 é ilustrado um ciclo de comunicação do FlexRay (ZURAWSKI, 2005).

Uma rede FlexRay é composta por um mínimo de dois nós, sendo que o ciclo de comunicação deve conter ao menos o segmento estático e o NIT. Um sistema de comunicação com múltiplos nós conectados através de ao menos um canal de comunicação é chamado de *cluster*.

Dois ou mais ciclos de comunicação podem formar um *ciclo de*

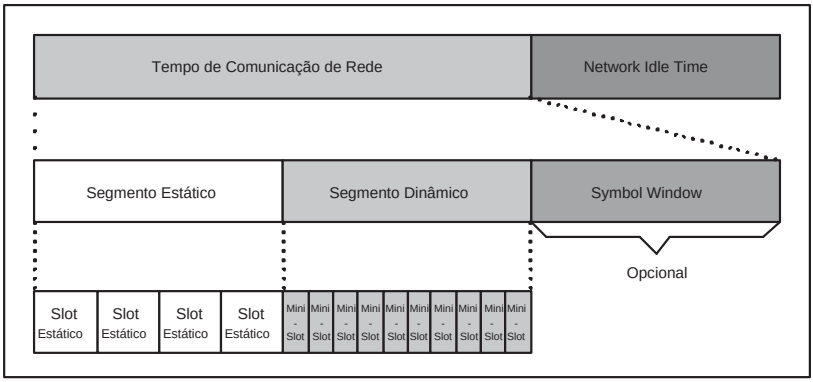

Figura 2.2: Ciclo de Comunicacao FlexRay (ZURAWSKI, 2005)

aplicação. No FlexRay, os ciclos de aplicação podem conter até 64 ciclos de comunicação, diferenciados com o uso de um contador global de 6 bits. Cada nó deve armazenar o contador localmente em *vCycleCounter* e seu valor atualizado é enviado no cabeçalho do quadro a cada transmissão efetuada (ZURAWSKI, 2005; NAVET; SIMONOT-LION, 2008).

2.2.1 Segmento Estático

O segmento estático de um ciclo de comunicação utiliza um método de controle de acesso ao meio baseado no *Time Division Multiple Access* (TDMA). Este segmento é composto por um número *gNumberOfStaticSlots* de *slots*, sendo que cada *slot* é formado por uma quantia idêntica de *macroticks* definidos em *gdStaticSlot*. Os *macroticks* são as unidades básicas de tempo em *cluster* FlexRay, e seu conceito será aprofundado na Seção 2.4.1. Em um *cluster* tanto *gNumberOfStaticSlots* como *gdStaticSlot* são valores globais constantes definidos em projeto.

Cada *slot* do segmento pertence exclusivamente a um controlador para a transmissão de um quadro, mas esta posse está relacionada apenas a um canal de comunicação. Como o FlexRay é um protocolo

que suporta até dois canais de comunicação podem surgir as seguintes combinações para um único nó no segmento estático: no *slot* 1 o nó transmite um quadro no canal A e um no canal B. No *slot* 2 transmite apenas um quadro no canal A[1]. No *slot* 3 não há transmissão de quadros. A Figura 2.3 ilustra estas possíveis combinações.

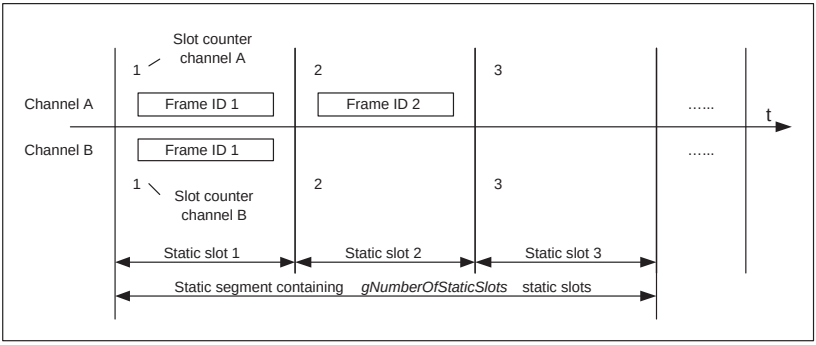

Figura 2.3: Estrutura do segmento Estático (FLEXRAY, 2005)

Na Figura 2.4 é apresentado outro exemplo de comunicação utilizando os dois canais do FlexRay. Aqui, os nós A e C utilizam os *slots* 1 e 3 em ambos os canais para a transmissão redundante de um quadro. O nó B está conectado apenas ao canal *b* e transmite seus quadros nos *slots* 2 e 4. O nó D também utiliza o *slot* 4 para transmissão, porém no canal a. Os nós C e E compartilham o *slot* 7 para a transmissão. No *slot* 5, o nó A transmite dois quadros diferentes. Os *slots* 6, 8, 9 e 10 não são utilizados (NAVET; SIMONOT-LION, 2008).

Para manter a ordem do escalonamento das mensagens, cada nó mantém contadores de *slots vSlotCounter* (um para cada canal). Estes contadores são iniciados com 1 no início de cada ciclo e incrementados simultaneamente no final de um *slot*.

Para cada nó de um *cluster* FlexRay um dos *slots* estáticos (definido pela constante *pKeySlotId*) pode ser designado para conter quadros do tipo especial *sync frame* relativos à sincronização e identificados por *pKeySlotUsedForSync*. Pode-se alocar *sync frames* específicos para

[1]De forma análoga, transmitir apenas no canal B também é permitido.

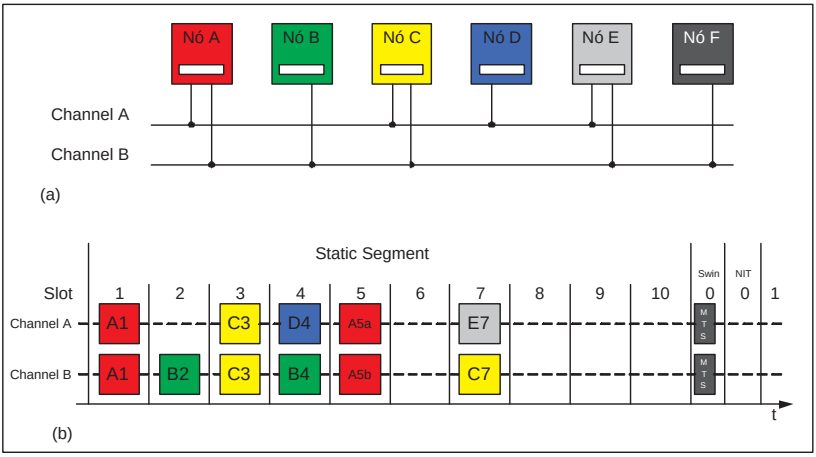

Figura 2.4: Comunicação no segmento estático. (a) *Cluster* FlexRay. (b) Transmissão de quadros em *slots* estáticos. (NAVET; SIMONOT-LION, 2008)

serem quadros de inicialização da rede, recebendo a identificação de *pKeySlotUsedForStartup*.

O segmento estático do FlexRay fornece um tempo de comunicação determinístico, já que se sabe exatamente quando um quadro é transmitido em determinado canal, dando assim uma grande garantia de latência. Esta garantia torna possível também a elaboração de projetos com reserva de banda, prevendo a integração futura de outros nós ao *cluster* (ZURAWSKI, 2005).

Além do determinismo, outra característica interessante do segmento estático é a propriedade de *composability* (NAVET; SIMONOT-LION, 2008). Em diversas áreas de engenharia, sistemas complexos são criados pela integração de subsistemas chamados componentes, que são bem definidos e testados. Em uma arquitetura dita *composable* esta integração deve ocorrer sem efeitos colaterais indesejáveis. Para que uma arquitetura seja *composable* devem ser obedecidos quatro princípios: desenvolvimento independente dos nós, estabilidade dos serviços já existentes no sistema, determinismo de réplica e *performability* do sistema de comunicação (KOPETZ; OBERMAISSER, 2002). A Figura 2.5

ilustra este comportamento. Neste exemplo, pode-se observar que a integração dos subsistemas do fornecedor 1, fornecedor 2 e fabricante não altera a posição e as características dos *slots* estáticos.

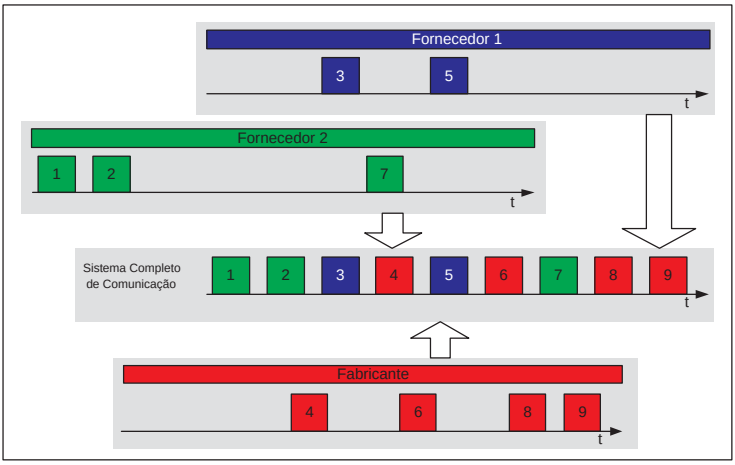

Figura 2.5: Exemplo de *composability* (NAVET; SIMONOT-LION, 2008)

A definição do escalonamento dos quadros do segmento estático no FlexRay deve ser feita na fase de projeto do sistema. Esta definição possui informações que são essenciais para os controladores, como por exemplo em qual *slot* enviarão ou receberão seus quadros.

2.2.2 Segmento Dinâmico

De forma semelhante ao segmento estático, o segmento dinâmico do FlexRay possui tamanho fixo e é dividido em *slots* que podem ser atribuídos a um controlador de comunicação para a transmissão de quadros. No entanto, o controle de acesso ao meio emprega um método mais flexível que utiliza os assim chamados *minislots*. Este método é chamado por alguns autores de *Flexible TDMA* (FTDMA) (NAVET; SIMONOT-LION, 2008).

Cada subdivisão do segmento dinâmico do FlexRay recebe o nome de *minislot* (MS). Os *minislots* possuem curta duração (segundo

a especificação, entre 2 e 378 μs) e seu número é definido por *gNumberOfMinislots*, sendo que cada um contém um número *gdMinislot* idêntico de *macroticks*. Tanto *gNumberOfMinislots* como *gdMinislot* são constantes globais para um dado *cluster* FlexRay. Na Figura 2.6(a) é ilustrado um ciclo de comunicação FlexRay, incluindo o segmento dinâmico e sua divisão em *minislots*.

Diferente dos *slots* do segmento estático que possuem tamanho fixo, a duração dos *slots* dinâmicos pode variar durante a execução do sistema e seu tamanho é especificado em múltiplos inteiros de MS. Este tamanho depende da transmissão ou não de quadros em determinado canal por um controlador, podendo resultar nas seguintes situações:

- Se não existe transmissão de dados o *slot* dinâmico possui o tamanho de um único MS.

- Se não existem MS em número suficiente para a transmissão completa do quadro, o quadro não é transmitido e o *slot* dinâmico tem o tamanho de um único MS.

- Se um quadro é transmitido, o *slot* dinâmico utiliza MS em número suficiente para acomodar a transmissão.

O comportamento acima descrito pode ser observado na Figura 2.6(b), onde no segmento dinâmico do primeiro ciclo o *slot* 1 transmite uma mensagem Ma que ocupa oito MS, o *slot* 2 transmite Mb que usa seis MS e o *slot* 3 não transmite nada e portanto gasta apenas um MS.

2.3 Formato do Quadro

Uma visão geral de um quadro FlexRay é dada na Figura 2.7. Como pode ser observado, é composto por três segmentos: cabeçalho ou *header*, carga ou *payload* e cauda ou *trailer*.

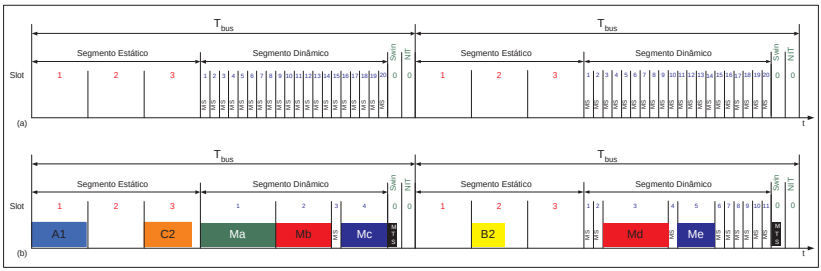

Figura 2.6: Exemplo de ciclo de comunicação FlexRay com segmento dinâmico. (a) Transmissão sem quadros. (b) Transmissão de quadros em um canal de comunicação

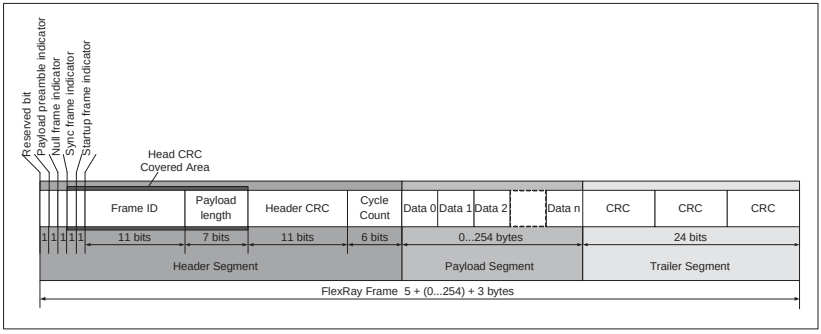

Figura 2.7: Formato de um quadro FlexRay (FLEXRAY, 2005)

2.3.1 Cabeçalho

O cabeçalho de um pacote FlexRay consiste de 5 bytes divididos em campos que contém informações definidas durante a fase de projeto. A função de cada campo é listada abaixo:

Reserved bit: bit reservado para uso futuro do protocolo. Deve ser ignorado pelos receptores, e mantido em nível lógico 0 pelos nós transmissores.

Payload preamble indicator: indica se o segmento de carga contém o vetor opcional para gerenciamento de rede ou o identificador de mensagem estendido.

Null frame indicator: utilizado para informar se o quadro

atual contém dados utilizáveis.

Sync frame indicator: informa se o quadro está sendo utilizado pelos controladores para sincronização global de relógios.

Startup frame indicator: bit responsável por sinalizar se o quadro está envolvido em um processo de inicialização do FlexRay.

Frame ID: define em qual *slot* o quadro deve ser transmitido. Um mesmo *frame ID* não é utilizado mais de uma vez em cada canal durante um ciclo de comunicação. Seu *range* é de 1 a 2047, e o ID 0 é considerado inválido.

Payload length: utilizado para indicar o tamanho do segmento de carga. O tamanho codificado neste campo equivale à metade do número de bytes de dados no segmento de carga, e pode variar de acordo com os seguintes fatores: quadros diferentes em um ciclo do segmento dinâmico e canal de comunicação utilizado.

Header CRC: contém um código de checagem de redundância cíclica que é computado sobre os campos *sync frame indicator*, *startup frame indicator*, *frame ID* e *payload length*.

Cycle count: indica no momento da transmissão do quadro o valor de *vCycleCounter* (contador local que armazena o número do ciclo) do nó transmissor.

2.3.2 Carga

O **segmento de carga** de um quadro FlexRay contém de 0 a 254 bytes, sendo estes os dados que estão sendo efetivamente trocados durante um ciclo de comunicação.

Para os quadros transmitidos no segmento dinâmico os dois primeiros bytes podem ser opcionalmente utilizados como um campo identificador de mensagem, permitindo aos nós receptores filtrar ou direcionar dados com base em seu conteúdo. O bit *Payload preamble indicator* do cabeçalho indica o uso ou não desta funcionalidade.

De forma similar ao segmento dinâmico, o segmento estático do FlexRay também permite o uso opcional de parte do *payload* para funções de gerenciamento. Neste caso, os doze primeiros bytes podem

ser utilizados como um vetor de gerenciamento de rede. O tamanho deste vetor é definido durante a fase de configuração do protocolo, e seu uso também é informado através do bit *Payload preamble indicator* (FLEXRAY, 2005).

2.3.3 *Trailer*

Um quadro FlexRay completo é protegido pelo *trailer*, que consiste de um CRC com 24 bits. Entretanto, partes do cabeçalho também são protegidos por um CRC adicional de 11 bits.

2.4 Representação do Tempo e Sincronização de Relógios

Em um sistema distribuído, cada dispositivo possui seu próprio relógio. Devido a vários fatores, como oscilações na temperatura ou diferenças entre os componentes que definem a frequência, os relógios de diferentes dispositivos irão divergir após algum tempo, mesmo se estes relógios houverem sido sincronizados em algum instante.

Em sistemas que utilizam redes com arbitragem do tipo *Time Division Multiple Access* (TDMA), é importante que os relógios dos dispositivos estejam, dentro de certas tolerâncias, sincronizados. O limite máximo para a diferença de valores entre dois relógios é chamado de *precisão*.

O protocolo de comunicação FlexRay utiliza um mecanismo de sincronização de relógios no qual cada nó da rede sincroniza a si mesmo em relação ao *cluster* com base na observação das transmissões feitas por outros nós.

2.4.1 Representação do tempo

A representação do tempo dentro de um nodo FlexRay é baseada em ciclos, *macroticks* (MT) e *microticks*, sendo que um *macrotick* é

composto por um número inteiro de *microticks*, e um ciclo é composto por um número inteiro de *macroticks*, conforme pode ser observado na Figura 2.8.

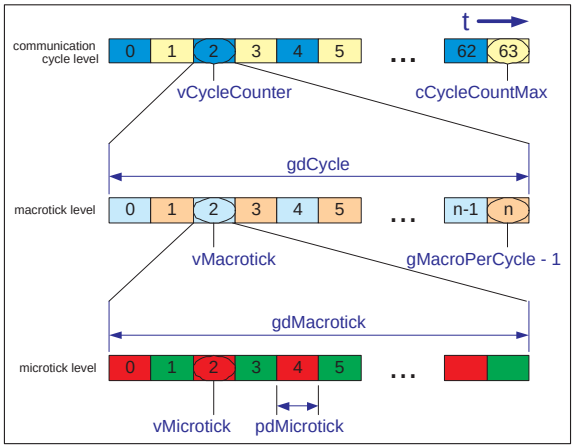

Figura 2.8: Representação do tempo no FlexRay (FLEXRAY, 2005)

Os *microticks* são unidades de tempo derivadas diretamente do relógio do controlador de comunicação de um nó FlexRay, e seu tamanho pode variar de um controlador para outro.

Já os *macroticks* são unidades de tempo que são sincronizadas dentro de um *cluster* FlexRay e, dentro das tolerâncias previstas pelo protocolo, tem a mesma duração para todos os nós de rede. A duração local de um *macrotick* é um inteiro de *microticks*, sendo que esta duração local pode variar de um nodo para outro, pois os *microticks* dependem do relógio local. Em cada nó, o valor local de *macroticks* é armazenado na variável *vMacrotick*, sendo que este valor vai de 0 a (*gMacroPerCycle*-1).

Por fim, um *ciclo* é composto por um número inteiro de *macroticks*. O número de macroticks por ciclo é fixo e deve ser idêntico para todos os nós em um *cluster*. O número do ciclo é armazenado localmente em um nó pela variável *vCycleCounter*, sendo que esta é incrementada no início de cada ciclo de comunicação. O valor de *vCy-*

cleCounter vai de 0 a *cCycleCountMax*. Quando *cCycleCountMax* é atingido, *vCycleCounter* é reiniciada para zero no início do ciclo de comunicação seguinte.

O *tempo global* de um *cluster* é o tempo utilizado por todos os dispositivos dentro deste *cluster*. Segundo a especificação, o FlexRay não possui um valor absoluto ou de referência para o tempo global, sendo que cada nó possui sua própria informação local para o tempo global.

O *tempo local* é o valor do relógio de um dispositivo específico no *cluster*, e é representado pelas variáveis *vCycleCount*, *vMacrotick* e *vMicrotick*. O tempo local é baseado na visão que o dispositivo tem do tempo global. Cada nó conectado à rede usa o algoritmo de sincronização do FlexRay para adaptar seu tempo local ao tempo global.

2.4.2 Processo de sincronização

A sincronização de relógios no FlexRay consiste de dois processos principais. O processo de geração de *macroticks* (*macrotick generation process*, MTG) controla os contadores de ciclo e *macroticks*, aplicando as correções de valores. O processo de sincronização de relógios (*clock synchronization process*, CSP) realiza a medição e armazenamento de valores de desvio e o cálculo do *offset* e valores de taxa de correção. Na Figura 2.9 são ilustrados os dois processos e sua relação com o MAC.

A tarefa primária da função de sincronização de relógios é garantir que as diferenças entre os nós da rede fiquem dentro da precisão definida para o *cluster*. O FlexRay utiliza para a correção de relógios um método que combina correção de *offset* e correção de taxa (taxa ou *rate*, também chamada correção de frequência). Existem condições que devem ser seguidas para a sincronização dos relógios dos nós em uma rede FlexRay:

- As correções de *offset* e *rate* devem ser feitas do mesmo modo em todos os nós.
- As correções de *offset* só podem ser realizadas durante o NIT e em

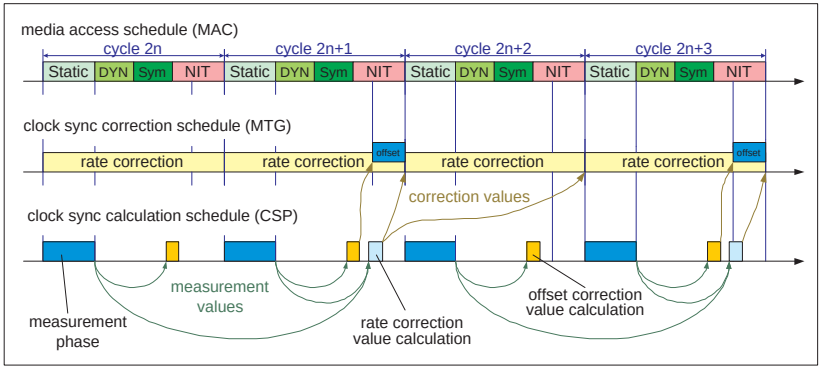

Figura 2.9: Relação temporal entre a sincronização de relógios, controle de acesso ao meio e execução de funções de sincronização de relógios (FLEXRAY, 2005)

ciclos com número ímpar. Deve iniciar no instante determinado pelo parâmetro *gOffsetCorrectionStart* e deve terminar antes do início do ciclo de comunicação seguinte.

- As modificações de *offset* são descritas pela variável *vOffsetCorrection*, onde é indicado o número de *microticks* que devem ser adicionados ao segmento de correção do NIT. O valor de *vOffsetCorrection* pode ser negativo e é calculado pelo algoritmo de sincronização de relógios. O cálculo de *vOffsetCorrection* é realizado a cada ciclo, mas as correções são aplicadas apenas no final de ciclos de comunicação com número ímpar.

- As correções de *rate* são descritas pela variável *vRateCorrection*, sendo um número inteiro de *microticks* (podendo ser negativo) que são adicionados ao número de *microticks* que foi configurado para cada ciclo de comunicação (*pMicroPerCycle*). O cálculo de *vRateCorrection* é feito pelo algoritmo de sincronização de relógios e inicia após o final do segmento estático de ciclos com número ímpar.

Os valores para a correção do *rate* e *offset* são calculados através de um algoritmo chamado *fault-tolerant midpoint algorithm* (FTM).

Após o cálculo dos valores pelo algoritmo, estes são verificados em relação aos valores limite definidos para as correções. Se os valores estiverem dentro dos limites estabelecidos, o relógio local é modificado de forma a sincronizar o nó ao relógio global. Esta modificação é feita pelo ajuste do número de *microticks* que compõe cada *macrotick*.

2.5 Controller Host Interface

O *controller host interface* (CHI) gerencia a troca de dados e fluxos de controle entre o processador *host* e o mecanismo do protocolo FlexRay dentro de cada nó.

O CHI contém dois blocos principais: *protocol data interface* e *message data interface*. A *protocol data interface* gerencia todas as trocas de dados relevantes à operação do protocolo, e a *message data interface* gerencia os dados relevantes às mensagens. A relação entre os blocos é ilustrada na Figura 2.10.

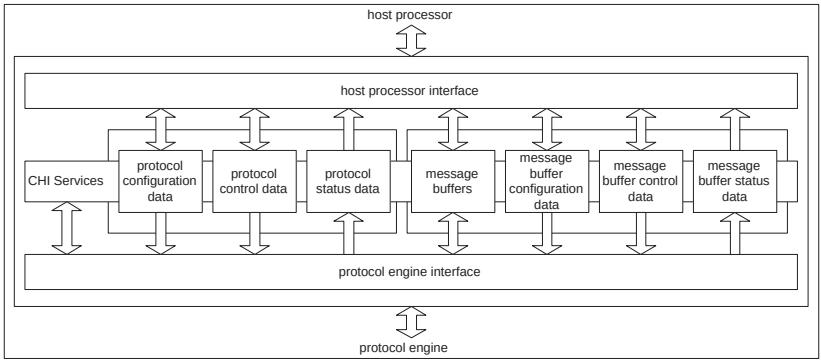

Figura 2.10: Arquitetura Conceitual do CHI (FLEXRAY, 2005)

A *protocol data interface* gerencia os dados de configuração do protocolo, os dados de controle do protocolo e os dados de *status* do protocolo. A *message data interface* gerencia os *buffers* de mensagens, os dados de configuração dos *buffers* e os dados de condições do *buffer*.

2.6 Emprego de Sistemas FlexRay

Os mecanismos de escalonamento de comunicações podem ser baseados em paradigmas diferentes, baseados no tempo ou na ocorrência de eventos no sistema (KESKIN, 2009).

Nas comunicações dirigidas por eventos (*event-triggered*, ET) as mensagens são geradas de acordo com a ocorrência de eventos no sistema, como por exemplo o acionamento de um sensor. Para garantir que a mensagem gerada seja consumida enquanto sua informação ainda é válida, é crucial que o protocolo associado garanta um tempo de resposta limitado para a transmissão. Um dos métodos para garantir o tempo de resposta é através de barramentos que utilizem arbitragens baseadas em prioridades atribuídas às mensagens (NAVET et al., 2005). Uma característica associada a comunicações dirigidas por eventos é o fato das mensagens frequentemente possuírem um caráter esporádico devido a sua forma de geração (SCHMIDT; SCHMIDT, 2008a).

Na abordagem onde as comunicações são dirigidas pelo tempo (*time-triggered*, TT), a comunicação entre os nós da rede é controlada pela passagem do tempo. Neste método de arbitragem de rede a transmissão de uma mensagem é realizada a intervalos pré-definidos que são definidos na fase de projeto do sistema, sendo o TDMA um método de arbitragem bastante conhecido para este tipo de rede. Como os intervalos de transmissão de mensagens são bem conhecidos, um sistema onde a comunicação é dirigida pelo tempo é determinista, mas em contrapartida a adição de novos nós ou mensagens no sistema certamente vai requerer um novo projeto devido a necessidade de alterações no escalonamento (NAVET et al., 2005).

Por seu caráter híbrido, o FlexRay suporta ambos os paradigmas de comunicação. Como o segmento estático do FlexRay opera com base no TDMA, este segmento é indicado para mensagens do tipo *time-triggered*. Já o segmento dinâmico é projetado para acomodar mensagens esporádicas (*event-triggered*) cuja geração ocorre em decorrência de eventos do sistema (SCHMIDT; SCHMIDT, 2008a; SCHMIDT;

SCHMIDT, 2008b; SCHEDL, 2007).

Como exemplo de uso do FlexRay em produtos comerciais pode-se citar a tecnologia *Adaptive Drive Chassis Control System* presente em veículos BMW (BMW Automobiles, 2010). Outro exemplo de uso comercial é o sistema de suspensão eletrônica dos veículos da série X5 da BMW, considerado o primeiro veículo comercial a utilizar sistemas FlexRay (Figura 2.11).

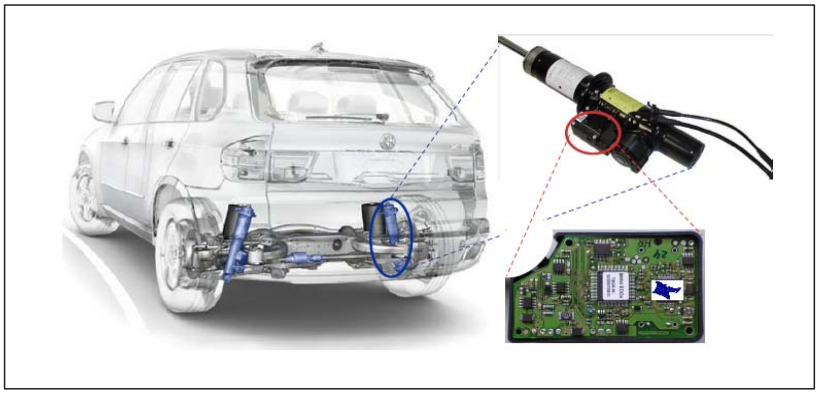

Figura 2.11: Suspensão eletrônica de veículos BMW X5 (SCHEDL, 2007)

Com o objetivo de ilustrar valores reais para os tamanhos do ciclo FlexRay, segmento estático e segmento dinâmico, são ilustrados na Tabela 2.1 os valores utilizados pela BMW para seus sistemas automotivos (SCHEDL, 2007).

Parâmetro	Valor
$gdCycle$	5 ms
ST_{bus}	3 ms
DYN_{bus}	2 ms
$gdMinislot$	6.875μs
$gNumberOfMinislots$	290 MS

Tabela 2.1: Parâmetros FlexRay adotados pela BMW (SCHEDL, 2007)

2.7 Lista de Parâmetros do FlexRay

Durante o projeto de um sistema FlexRay devem ser feitas especificações para dezenas de parâmetros de baixo nível que definem o comportamento da rede e que devem ser idênticos para todos os controladores de comunicação participantes de um dado *cluster*. Nas Tabelas 2.2 e 2.3 são apresentados alguns destes parâmetros, inclusive os que serão citados durante este trabalho.

A lista completa dos parâmetros relacionados ao Protocolo de Comunicação FlexRay pode ser encontrada em (FLEXRAY, 2005).

2.8 Conclusões

O FlexRay é um sistema de comunicação altamente divulgado como sendo o futuro padrão *de fato* para aplicações automotivas (POP et al., 2008). Sua especificação define dois canais de comunicação operando a velocidades de até 10Mbit/s. Sua operação é baseada em um ciclo de comunicação que inclui um segmento estático que utiliza uma arbitragem de barramento do tipo TDMA e um segmento dinâmico cuja arbitragem de barramento é baseada no FTDMA.

Este capítulo apresentou as principais características relacionadas à especificação do protocolo, com base no *FlexRay Protocol Specification V2.1 Rev.A*. No próximo capítulo serão apresentadas as soluções relacionadas à análise de escalonabilidade para ambos os segmentos do FlexRay disponíveis na literatura.

Parâmetro	Descrição	Valor
$cCycleCountMax$	Número máximo de ciclos em um *cluster* de nós de rede	63
$cdCycleMax$	Tamanho máximo de um ciclo	16000μs
$cdMaxMTNom$	Duração máxima de um macrotick	6μs
$cdMinMTNom$	Duração mínima de um macrotick	1μs
$cPayloadLengthMax$	Tamanho máximo do segmento de dados de um quadro	127 two-byte words
$cStaticSlotIDMax$	Maior número de identificação de um *slot* estático	1024
$gdMinislot$	Duração de um MS	2-63 MT
$gdStaticSlot$	Duração de um *slot* estático	4-661 MT
$gMacroPerCycle$	Número de *macrotiks* em um ciclo	10 - 16000 MT
$gNumberOfMinislots$	Número de MS no segmento dinâmico	0-7986
$gNumberOfStaticSlots$	Número de *slots* no segmento estático	2-cStatic SlotIDMax
$pChannels$	Canais utilizados pelo *cluster*	[A, B, A&B]
$gdCycle$	Tamanho do ciclo em μs	10μs - cdCycleMax
$gdMacrotick$	Duração de um macrotick em μs	1-6μs
$pLatestTx$	Número do último MS no qual a transmissão de um quadro dinâmico pode iniciar	0-7980 MS

Tabela 2.2: Parâmetros do FlexRay (FLEXRAY, 2005)

Parâmetro	Descrição	Valor
$gdSymbolWindow$	Duração de *Symbol Window*	0-142 MT
$gOffsetCorrectionStart$	Início da fase de correção de *offset* dentro do NIT, expresso em *microticks* a partir do início do ciclo	9-15999 MT
$gPayloadLengthStatic$	Tamanho da carga em um quadro estático	0-$cPayload LenghMax$ two-byte words
$gAssumedPrecision$	Precisão assumida para uma rede	0.15-11.7μs
$gdbit$	Tempo nominal de um bit um quadro	$cSamples PerBit$* $gdSampleClock period[\mu s]$
$cSamplesPerBit$	Número de amostras utilizadas para a determinação do valor de um bit	8
$gdNIT$	Duração do NIT	2-805 MT
$gOffsetCorrectionMax$	Magnitude global no *cluster* do necessário máximo valor para a correção do *offset*	0.15 - 383.567 μs
$pMicroPerCycle$	Número nominal local de *microtiks* no ciclo de comunicação em um nó	640 - 640000 μT

Tabela 2.3: Parâmetros do FlexRay - continuação (FLEXRAY, 2005)

Capítulo 3

Análise de Escalonabilidade para FlexRay

Este capítulo apresenta os trabalhos da literatura que de alguma forma se relacionam com a análise de escalonabilidade em sistemas que utilizam barramentos de comunicação tipo FlexRay.

O material disponível pode ser dividido em dois grupos, um relacionado a trabalhos cujo foco é a definição de parâmetros de sistemas que utilizam o FlexRay e outro relacionado a trabalhos que abordam a análise de escalonabilidade de mensagens em ambos os segmentos.

Os trabalhos que abordam a definição de parâmetros utilizam como base o conjunto de mensagens que devem ser alocadas em um dos segmentos, e a partir deste conjunto definem os valores para diversos parâmetros como tamanho dos *slots* e ciclos, alocando também os $FrameIDs$ para cada mensagem. Uma característica relacionada a este tipo de solução é que os valores resultantes em geral são ótimos para o conjunto de mensagens dado como entrada, sendo que por "ótimos" deve-se entender valores que são exatamente os necessários para

o conjunto de mensagens relacionado ser escalonável. Entretanto, um sistema com valores ótimos pode se mostrar problemático na prática, pois qualquer modificação (como a adição de um novo dispositivo ou mudanças na característica de uma mensagem) poderá implicar na necessidade de geração de um novo conjunto completo de valores. Para ilustrar este problema, considere um veículo em que se deseja alterar apenas um dos subsistemas. Dependendo da alteração a ser realizada, em um sistema projetado para ser ótimo poderá ser necessária a reconfiguração de todos os dispositivos ligados à rede, sendo que em um sistema projetado com alguma folga seria necessário alterar apenas os dispositivos relacionados ao subsistema em questão.

Os trabalhos relacionados à análise de escalonabilidade no FlexRay apresentam propostas de métodos para o cálculo do tempo de resposta das mensagens, sendo que este tempo de resposta é utilizado para definir se todas as mensagens conseguem cumprir seus *deadlines*, indicando assim se o sistema é ou não escalonável.

Este capítulo está organizado como se segue: A título de informação, a Seção 3.1 apresenta os trabalhos relacionados ao segmento estático. Mas como o foco desta dissertação é o segmento dinâmico, estes são apresentados de forma resumida. As Seções 3.2.1, 3.2.2, 3.2.3 e 3.2.4 descrevem os trabalhos relacionados ao segmento dinâmico do FlexRay. Em cada seção é feita uma descrição do trabalho, apresentado o modelo de sistema e o método proposto, sendo também feitas considerações sobre os mesmos.

3.1 Segmento Estático

Segundo (POP et al., 2008), a análise do tempo de resposta no segmento estático do FlexRay pode ser reduzida ao problema de análise do tempo de resposta em um protocolo do tipo TDMA, de forma que esta seção irá abordar de forma breve este assunto.

Como o controlador de comunicação de um dispositivo FlexRay decide qual mensagem será enviada para o barramento em um deter-

minado *slot* no início deste *slot*, o pior caso para uma mensagem m é ser gerada imediatamente após o início do *slot* correspondente ao seu *FrameID*, pois isto fará com que m tenha que esperar até o início do próximo ciclo para competir pelo barramento (POP et al., 2008).

Como já discutido no Capítulo 2, o segmento estático utiliza uma arbitragem baseada no TDMA, sendo o segmento é dividido em *slots* de tamanho fixo e idêntico que se repetem a um intervalo fixo de tempo. Devido a este esquema de arbitragem, uma mensagem deste segmento que perca o início do seu *slot* não irá competir pelo barramento no início do próximo ciclo e sim esperar um ciclo FlexRay completo, até o início do próximo *slot* associado ao seu *FrameID*. Isto pode ser observado na Figura 3.1, na qual é ilustrado o tempo de reposta para uma mensagem m associada ao *slot* 2 do segmento estático.

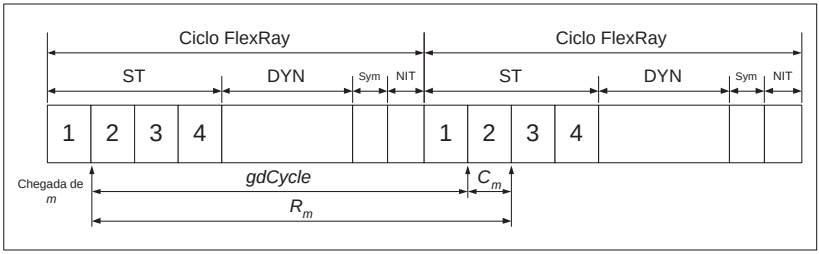

Figura 3.1: Tempo de Resposta para uma Mensagem Estática

O tempo de resposta R_m (em μs) para uma mensagem estática m a partir do controlador de comunicação de um dispositivo pode ser calculado pela Equação 3.1, em que C_m é o tempo necessário para a transmissão da mensagem em μs, dado pela Equação 3.2. Na Equação 3.2 *bus_speed* é a velocidade do barramento dada em $Mbit/s$.

$$R_m = gdCycle + C_m \tag{3.1}$$

$$C_m = \frac{(5bytes + gPayloadLengthStatic + 3bytes) * 8}{bus\_speed} \tag{3.2}$$

Um método para a análise do tempo de resposta de uma mensagem estática desde sua geração até seu consumo é proposto em (GUERMAZI; GEORGE, 2008), em que se considera uma arquitetura AUTOSAR[1] para o dispositivo, de forma que o módulo de comunicação é composto por um *driver* FlexRay, uma *interface* FlexRay e um roteador para os pacotes de dados (*PDU-Router*). Entretanto, o método proposto utiliza uma equação similar a Equação 3.1 como componente da equação final para o cálculo do tempo de resposta fim-a-fim.

Na literatura sobre o FlexRay também existem outros trabalhos que abordam diferentes aspectos relacionados ao escalonamento de mensagens no segmento estático. Estes trabalhos serão listados abaixo.

***Message Scheduling for the FlexRay Protocol: The Static Segment* (SCHMIDT; SCHMIDT, 2008b)**: Este trabalho apresenta um método que utiliza Programação Linear Inteira (*Integer Linear Programming*, ILP) para obter as definições de um sistema onde está presente apenas o segmento estático. Estas definições incluem valores para $gdStaticSlot$ e $gNumberOfStaticSlots$ e a atribuição dos *FrameID* para as mensagens. O trabalho propõe também que as mensagens geradas por processos diferentes dentro de um mesmo nó sejam empacotadas em uma mensagem maior única com o objetivo de maximizar a utilização da rede.

***An Effective GA-based Scheduling Algorithm for FlexRay Systems* (DING et al., 2008):** Este trabalho propõe um método para o escalonamento das mensagens do segmento estático com o uso de algoritmos genéticos.

***Configuring the Communication on FlexRay - The Case of the Static Segment* (GRENIER et al., 2008):** Os autores propõe o uso de heurísticas para obter uma escala otimizada das mensagens do segmento estático de um sistema. Estas heurísticas assumem vá-

[1] O AUTOSAR (*AUTomotive Open System ARchitecture*) é uma metodologia para desenvolvimento de dispositivos automotivos proposta por um conjunto de fabricantes de veículos e define camadas para a arquitetura de software, sendo que cada camada é composta por componentes que se comunicam através de *interfaces* padronizadas. Mais informações sobre o AUTOSAR podem ser encontradas em (AUTOSAR, 2010).

rias restrições, sendo que estas, segundo os autores do trabalho, são similares às utilizadas pela BMW para veículos nos quais o FlexRay é utilizado.

***Scheduling the FlexRay Bus Using Optimization Techniques* (ZENG et al., 2009):** Neste trabalho é apresentado um método que utiliza ILP para definir o escalonamento de mensagens estáticas considerando o padrão OSEK/VDX. O OSEK/VDX é um projeto conjunto de fabricantes de veículos que visa desenvolver um padrão aberto para arquiteturas de sistemas distribuídos em veículos. Mais informações sobre o OSEK/VDX podem ser encontradas em (OSEK VDX, 2010).

3.2 Segmento Dinâmico

Como descrito no início deste capítulo, o foco desta dissertação é o segmento dinâmico do FlexRay. Por isso, serão apresentados a seguir os trabalhos da literatura que abordam a análise de escalonabilidade neste segmento.

3.2.1 Artigo 1: "*Message Scheduling for the FlexRay Protocol: The Dynamic Segment*"

O artigo *Message Scheduling for the FlexRay Protocol: The Dynamic Segment* (SCHMIDT; SCHMIDT, 2008a) apresenta um método que, dados um conjunto de mensagens dinâmicas periódicas ou esporádicas e o tamanho do segmento estático, define os parâmetros para um sistema FlexRay. Uma das principais características do trabalho é a proposta do uso de agrupamento de mensagens e reserva de recursos como meio para se otimizar a utilização da largura de banda, sendo utilizada Programação Linear Inteira ILP para selecionar o conjunto de parâmetros que permitem a construção da escala viável de mensagens mais eficiente para o barramento.

Diferente do que será feito para demais trabalhos relacionados ao segmento dinâmico do FlexRay esta proposta não será analisada de

forma aprofundada, pois não é de grande interesse para esta dissertação devido a dois motivos: o primeiro é que, como o método proposto em (SCHMIDT; SCHMIDT, 2008a) procura fornecer parâmetros ótimos para um sistema, o mesmo se encaixa na categoria de trabalhos que, como já discutido na introdução deste capítulo, oferecem soluções pouco flexíveis em relação a alterações futuras. O segundo motivo é o fato de que o foco principal desta dissertação é a análise do tempo de resposta de mensagens dinâmicas, mas no método proposto no artigo em questão esta análise é feita de forma indireta, sendo as informações sobre período e *deadline* de mensagens utilizadas em conjunto com vários outros parâmetros na formulação ILP.

3.2.2 Artigo 2: "*Timing Analysis of the FlexRay Communication Protocol*"

O trabalho intitulado "*Timing Analysis of the FlexRay Communication Protocol*" (POP et al., 2008) é considerado por alguns autores a primeira tentativa de modelar formalmente o segmento dinâmico do FlexRay para fins de análise (HAGIESCU et al., 2007). O trabalho apresenta dois métodos para obtenção do maior atraso que uma mensagem m pode sofrer devido a mensagens de prioridade mais alta, sendo que este atraso é utilizado para calcular o pior caso para o tempo de reposta R_m. O trabalho também apresenta dois algoritmos que utilizam o tempo de resposta calculado para a definição dos parâmetros do sistema. Como o foco desta dissertação é a análise do tempo de resposta de mensagens do segmento dinâmico do FlexRay, apenas os dois métodos propostos para o cálculo de R_m serão discutidos.

3.2.2.1 Modelo de Sistema

O artigo considera arquiteturas que consistem de nós interligados por um canal de comunicação FlexRay. Cada nó é composto por dois componentes principais: uma Unidade Central de Processamento (*Central Processing Unit*, CPU) e um controlador de comunicação, os

quais são interconectados através de um CHI. O controlador executa seus serviços de forma independente da CPU e implementa o protocolo FlexRay. Cada nó possui um *kernel* de tempo real que contém dois escalonadores, um baseado em executivo cíclico (*Static Cyclic Scheduling*, SCS) e outro baseado em prioridades (*Fixed Priority Scheduling*, FPS). Uma tarefa entra na fila de aptos após a chegada de todas as suas entradas e disponibiliza os dados de saída após seu término. Uma mensagem entra na fila de aptos após a conclusão da tarefa de envio associada e se torna disponível para o receptor após o final de sua transmissão. O tempo de comunicação entre tarefas em um mesmo nó é considerado parte do tempo de resposta da tarefa e não é modelado explicitamente.

É considerado que a política de escalonamento para cada tarefa é conhecida (SCS ou FPS). O modelo considera tanto mensagens periódicas como esporádicas, sendo que mensagens TT são alocadas no segmento estático, e mensagens ET são alocadas no segmento dinâmico. Cada mensagem ativada por eventos tem uma prioridade definida.

O tamanho de cada mensagem m é fornecido, sendo diretamente convertido em tempo de comunicação C_m através da Equação 3.3, na qual *bus_speed* é a velocidade do barramento em $bits/s$.

$$C_m = Frame\_Size(m)/bus\_speed \tag{3.3}$$

3.2.2.2 Método Proposto

Um dos objetivos do trabalho em (POP et al., 2008) é apresentar uma maneira de analisar o tempo de resposta no pior caso (*worst-case response time*, wcrt) de mensagens ET alocadas no segmento dinâmico do FlexRay. Segundo os autores, o wcrt R_m de uma mensagem m é dado pela Equação 3.4, na qual C_m é dado pela Equação 3.3 e t é um dado intervalo de análise.

$$R_m(t) = \sigma_m + w_m(t) + C_m \tag{3.4}$$

O valor de σ_m vem do fato de que o controlador de comunica-

ção decide se uma mensagem será transmitida pelo barramento em um determinado *slot* no início deste *slot*. Como consequência, no pior caso uma mensagem m é gerada pela tarefa emissora imediatamente após o início do *slot* associado ao seu $FrameID$, de forma que m deve esperar até o início do próximo ciclo para que possa competir pelo barramento. Este atraso é dado pela Equação 3.5 [2]:

$$\sigma_m = gdCycle - (ST_{bus} + (FrameID_m - 1) \times gdMinislot). \quad (3.5)$$

Para que R_m possa ser calculado resta definir o máximo atraso w_m gerado por repetições do segmento estático e por mensagens dinâmicas de maior prioridade que m. Segundo os autores, uma mensagem dinâmica m que está na fila de aptos pode ser atrasada por:

- Mensagens no mesmo nó que utilizem o mesmo $FrameID$ que m mas que tenham prioridade mais alta. O conjunto destas mensagens é denominado $hp(m)$.
- Qualquer mensagem do sistema que tenha um identificador menor que m (ou seja, prioridade mais alta). O conjunto destas mensagens é denominado $lf(m)$.
- Minislots dinâmicos não utilizados e com identificador mais baixo que m. Apesar de não utilizados, estes *minislots* atrasam a transmissão de m por um intervalo de tempo igual a $gdMinislot$ por *minislot*. O conjunto destes *minislots* é denominado $ms(m)$.

Segundo os autores da proposta, o atraso w_m pode ser estendido por mais de um ciclo, como apresentado na Equação 3.6:

$$w_m(t) = BusCycles_m(t) \times gdCycle + w'_m(t) \quad (3.6)$$

em que $BusCycles_m(t)$ é o número de ciclos nos quais a transmissão

[2] Na Equação original apresentada em (POP et al., 2008), o tamanho de um ciclo é representado pela expressão T_{bus}. No entanto, com o objetivo de padronizar as expressões presentes neste trabalho T_{bus} foi substituído por $gdCycle$, pois esta é a expressão da especificação do FlexRay para o tamanho de um ciclo.

de m não é possível devido a $hp(m)$, $lp(m)$ e $ms(m)$. O atraso $w'_m(t)$ representa o tempo decorrido no último ciclo até o início da transmissão de m, sendo considerado a partir do início daquele ciclo.

Como os termos da Equação 3.6 são funções do tempo deve-se considerar todos os elementos de $hp(m)$, $lp(m)$ e $ms(m)$ que podem ocorrer durante um dado intervalo t. Para isso, considera-se o conjunto $hp(m,t)$ que contém todas as possíveis ocorrências dos elementos de $hp(m)$ no intervalo t. O número de ocorrências de uma mensagem $l \in hp(m)$ é igual a $\lceil (J_l + t)/T_l \rceil$, sendo T_l o período da mensagem l e J_l o *jitter*. De forma similar, $lf(m,t)$ e $ms(m,t)$ consideram todas as ocorrências no intervalo t dos elementos de $lf(m)$ e $ms(m)$ respectivamente.

Para o cálculo de $BusCycles_m(t)$ e $w'_m(t)$ o trabalho em (POP et al., 2008) apresenta duas soluções, uma ótima e outra heurística, sendo que as mesmas serão reproduzidas a seguir.

Solução Exata para $BusCycles_m(t)$

A solução exata para o cálculo de $BusCycles_m(t)$ parte da observação de que uma mensagem m com $FrameID_m$ não pode ser enviada por um nó N_p durante um ciclo de barramento b se pelo menos uma das seguintes condições for verdadeira:

1. Existe muita interferência de elementos em $lf(m)$ e $ms(m)$ de forma que o contador de *slots* excede o valor de $pLatestTx_{N_p}$, tornando impossível para N_p iniciar a transmissão de m durante o ciclo b.

2. O *slot* dinâmico com $FrameID = m$ em b é utilizado por outra mensagem de $hp(m)$.

Quando um ciclo satisfaz ao menos uma das condições acima ele é chamado de "cheio" ou "'completo"', pois não pode ser utilizado para a transmissão da mensagem m em análise. No pior caso, o valor de $BusCycles_m(t)$ é portanto o máximo número de ciclos que podem ser preenchidos utilizando-se elementos de $hp(m)$, $lp(m)$ e $ms(m)$.

As condições descritas acima implicam que no pior caso cada ciclo "cheio" irá conter apenas mensagens de $lf(m)$ ou apenas uma mensagem de $hp(m)$. Isso significa que o atraso produzido por elementos de $lf(m,t)$ se soma ao atraso produzido por elementos de $hp(m,t)$:

$$BusCycles_m(t) = BusCycles_m(hp(m,t)) + BusCycles_m(lf(m,t), ms(m,t)) \quad (3.7)$$

em que $BusCycles_m(hp(m,t))$ é o número de ciclos nos quais o atraso é causado por mensagens em $hp(m,t)$ e $BusCycles_m(lf(m,t), ms(m,t))$ é o número de ciclos "cheios" nos quais a transmissão de m é impossível devido a elementos em $lf(m,t)$ e $ms(m,t)$.

Como cada mensagem em $hp(m,t)$ atrasa a transmissão de m por um ciclo de comunicação, a ocorrência de mensagens de $hp(m)$ durante o intervalo t irá produzir um atraso igual ao número total de elementos em $hp(m,t)$:

$$BusCycles_m(hp(m,t)) = |hp(m,t)| . \quad (3.8)$$

O problema que ainda precisa ser resolvido é determinar quantos ciclos de comunicação podem ser preenchidos de acordo com a primeira condição. Os autores propõe utilizar uma formulação ILP visando obter uma solução exata. Esta formulação parte da observação de que considerando-se a existência de n elementos em $lf(m)$, existem ao menos n ciclos de comunicação que podem ser preenchidos. Para cada um destes ciclos é criada uma variável $y_{i=1...n}$ que é definida como 1 se o i-ésimo ciclo é preenchido com elementos de $lf(m,t)$ e $ms(m,t)$, e 0 se não é preenchido (ou seja, se há espaço para a transmissão da mensagem m sob análise).

O objetivo do problema ILP é maximizar o número de ciclos de comunicação cheios:

$$BusCycles_m(lf(m,t), ms(m,t)) = \sum_{i=1..n} yi \quad (3.9)$$

O objetivo está sujeito a um conjunto de condições que definem y_i como 1 ou 0. É alocada uma variável binária x_{ijk}, definida como 1 se uma mensagem $m_k \in lf(m,t)(k = 1...n)$ é enviada durante o ciclo i com um $FrameIDj = 1...FrameID_m$. A carga transmitida em cada ciclo de comunicação pode ser expressa como:

$$Load_i = \sum_{\substack{m_k \in lf(m,t) \\ j=1...FrameID_m}} x_{ijk} \times C_k + \sum_{j=1...FrameID_m} \left(1 - \sum_{m_k \in lf(m,t)} x_{ijk}\right) \times gdMinislot, \quad (3.10)$$

em que C_k é o tempo de comunicação de uma mensagem $m_k \in lf(m,t)$ (Equação 3.3). Cada termo do somatório da equação captura as particularidades de quadros dinâmicos do FlexRay: se uma mensagem k é transmitida em no ciclo i com identificador j, então $x_{ijk} = 1$ e o tamanho do quadro a ser transmitido é igual ao tamanho da mensagem k. Se $x_{ijk} = 0$ não existe a transmissão do quadro, mas ainda existe um atraso causado pelo *minislot* não utilizado.

A condição que define cada variável y_i como 1 é:

$$Load_i > pLatestTx_m \times gdMinislot \times y_i. \quad (3.11)$$

Esta condição reforça que uma variável y_i não pode ser definida como 1 a não ser que a interferência total causada por elementos de $lp(m,t)$ e $ms(m,t)$ no ciclo i excedam o $pLatestTx_m$ da mensagem m sob análise.

Além da condição expressa na Equação 3.11, outras condições para o modelo são:

- cada mensagem $m_k \in lf(m,t)$ é enviada em apenas um ciclo i:

$$\sum_{\substack{m_k \in lf(m,t) \\ j=1...FrameID_m}} x_{ijk} \leq 1, \forall m_k \in lf(m) \quad (3.12)$$

- cada identificador de quadro é utilizado apenas uma vez em cada ciclo:

$$\sum_{k=1...n} x_{ijk} \leq 1, \forall i, j \tag{3.13}$$

- cada mensagem $m_k \in lf(m,t)$ é transmitida utilizando seu identificador:

$$x_{ijk} \leq Frame_{jk}, \forall i, j, k \tag{3.14}$$

sendo $Frame_{jk}$ uma constante binária que possui com valor 1 se a mensagem $m_k \in lf(m,t)$ possui um identificador $FrameID_{m_k} = j$ ou 0 caso contrário.

Finalmente, deve-se reforçar que em cada ciclo i nenhuma mensagem m_k irá iniciar sua transmissão após seu $pLatestTx_{m_k}$:

$$\sum_{\substack{m_q \in lf(m,t) \\ p=1...j-1}} x_{ipq} \times C_q + \sum_{p=1...j-1} \left(1 - \sum_{m_q \in lf(m,t)} x_{ipq}\right) \times gdMinislot \leq pLatestTx_k \times gdMinislot \tag{3.15}$$

As condições expressas pelas Equações 3.10-3.15 em conjunto com a função objetivo expressa na Equação 3.9 definem uma formulação para ILP que determina o máximo número de ciclos que podem ser preenchidos com elementos de $lf(m,t)$ e $ms(ms,t)$. Adicionando o resultado obtido com o valor definido na Equação 3.8 obtém-se o número total de $BusCycles_m(t)$ (Equação 3.7).

Solução Exata para $w'_m(t)$

O problema de definir o valor para w'_m equivale a definir qual, no pior caso, a máxima carga possível no primeiro ciclo que não é completamente preenchido por elementos de $lf(m,t)$ e $ms(m,t)$ (ou seja, o ciclo que não satisfaz a condição representada na Equação 3.11). Este valor pode ser determinado pela mesma formulação ILP apresentada acima e utilizada para determinar o valor de $BusCycles_m(lf(m,t), ms(m,t))$, sendo feitas as seguintes modificações:

- Como o valor de $BusCycles_m$ é conhecido são adicionadas condições que forçam os valores $y_i = 1$ para todo $i = 1...BusCycles_m$ e $y_i = 0$ para cada $i = BusCycles_m+1...n$. Desta forma, as mensagens serão agrupadas de forma que os ciclos de 1 até $BusCycles_m$ são preenchidos (satisfazendo a condição expressa na Equação 3.11), enquanto os ciclos restantes são mantidos não preenchidos.

- Utilizando o mesmo conjunto de condições 3.10-3.15 para preencher os primeiros $BusCycles_m$ ciclos, o objetivo descrito na Equação 3.9 é substituído por outra expressando que a carga do ciclo $BusCycles_m + 1$ deve ser maximizada:

$$maximize\ Load_L,\ \text{for}\ L = BusCycles_m + 1 \tag{3.16}$$

Soluções Heurísticas para $BusCycles_m$ e w'_m

As soluções apresentadas para determinar valores exatos para $BusCycles_m$ e w'_m são, segundo os próprios autores, inaceitáveis na prática devido à complexidade computacional necessária para executar os complexos algoritmos envolvidos. Por isso, o trabalho em (POP et al., 2008) também apresenta soluções heurísticas para determinar $BusCycles_m$ e w'_m.

A primeira solução heurística é utilizada para computar o valor de $BusCycles_m$, e parte da observação de que, em um ciclo no qual uma mensagem m é enviada por um nó N_p durante um *slot* dinâmico com

$FrameID_m$ no pior caso existirão no máximo $FrameID_m - 1$ *minislots* não utilizados antes de m ser transmitida. O atraso produzido por estes minislots passa a ser considerado parte do tempo de comunicação da mensagem, conforme a Equação 3.17:

$$C'_m = (FrameID_m - 1) \times gdMinislot + C_m \tag{3.17}$$

Segundo os autores, como a duração de um *minislot* tem uma ordem de magnitude pequena se comparada ao tamanho de um ciclo, a aproximação não introduzirá um pessimismo significante no resultado final. Entretanto, esta afirmação será discutida na Seção 3.2.2.3.

Após esta simplificação, o problema que resta para resolver é quantos ciclos podem ser completados com os elementos do conjunto $lf'(m,t)$, que consiste de todos os elementos de $lf(m,t)$ para os quais são computados tempos de comunicação utilizando-se a Equação 3.17. Se forem ignoradas as condições utilizadas no método exato, determinar $BusCycles(lf'(m,t))$ se torna um problema de empacotamento (*bin packing problem*). No problema do *bin packing* se procura maximizar o número de caixas que podem ser preenchidas até uma capacidade mínima usando um dado conjunto de itens. Na heurística proposta, as mensagens em $lf'(m,t)$ são os itens, o segmento dinâmico são as caixas e $pLatestTx_m \times gdMinislot$ é a capacidade mínima requerida para encher uma caixa. A solução para o problema é dada por uma abordagem apresentada em (Labbe, M. and Laporte, G. and Martello, S., 1995) que é descrita de forma resumida a seguir.

Em primeiro lugar, organiza-se as mensagens por ordem decrescente em relação ao seu tamanho, de forma que $C'_1 > C'_2 > C'_n$. Considerando-se a análise de uma mensagem m, antes de se computar um limite superior para $BusCycles(lf'(m,t))$ o conjunto $lf'(m,t)$ é reduzido de acordo com os seguintes critérios:

1. Qualquer mensagem j com $C'_j > pLatestTx_m$ pode preencher sozinha um segmento dinâmico do FlexRay. Denomina-se $R1$ o número destes itens e os mesmos são removidos de $lf'(m,t)$.

2. Se duas mensagens k e l satisfazem $C'_k + C'_l = pLatestTx_m + 1$ então existe uma solução ótima na qual um segmento dinâmico do FlexRay contém apenas as mensagens k e l. O número de segmentos dinâmicos que podem ser utilizados desta forma é denomidado $R2$ e as mensagens que satisfazem o critério de redução são removidas de $lf'(m,t)$.

3. Seja k a mensagem com maior prioridade tal que $C'_1 + \sum_{j=k}^{n} C'_j > pLatestTx_m$. Se $C'_1 + C'_k \geq pLatestTx_m$ então existe uma solução ótima na qual um segmento dinâmico do FlexRay contém apenas as mensagens 1 e k. O número de segmentos dinâmicos que podem ser utilizados desta forma é denomidado $R3$ e as mensagens que satisfazem o critério de redução são removidas de $lf'(m,t)$.

No segundo passo, as n' mensagens restantes em $lf'(m,t)$ são utilizadas para preencher segmentos dinâmicos com um valor mínimo de $pLatestTx_m + 1$. As heurísticas apresentadas abaixo determinam um limite superior para o valor desejado.

1. Como não existe uma dupla das mensagens restantes que possa preencher um segmento dinâmico até o valor mínimo, então $U0 = \left\lfloor \frac{n'}{2} \right\rfloor$.

2. O relaxamento contínuo do problema dá outro limite: $U1 = \left\lfloor \sum_{j=1}^{n'} \frac{C'_j}{pLatestTx_m+1} \right\rfloor$.

3. Seja $t = \min\{s : \sum_{h=s}^{n'} C'_h < pLatestTx_m + 1\}$ e defina-se que $p(j) = \min\{p : \sum_{h=j}^{j+p} C'_h \geq pLatestTx_m + 1\}$ para $j = 1, .., \tau = \min(\left\lfloor \frac{n'}{2} \right\rfloor, t-1)$. Então para $k = 0, 1, .., \tau$ o valor de $U2(k)$ é dado por $U2(k) = k + \left\lfloor \sum_{j=k+1}^{n'-\alpha(k)} \frac{C'_j}{pLatestTx_m} \right\rfloor$, sendo $\alpha(0)$ e $\alpha(k) = \sum_{j=1}^{k} p(j)$ para $k > 0$ é um limite superior válido se considerado o conjunto reduzido de mensagens. Denomina-se $U2$ o mínimo dos valores: $U2 = \min(U2(k)), k = 1, ..., \tau$.

4. Defina-se $q(j) = \beta(j)$ se $\beta(j) > j$, $q(j) = \beta(j) + 1$ caso contrário, e seja $\beta(j)$ o menor índice de mensagem k tal que $C'_j +$

$\sum_{k=1,k\neq j}^{\beta(j)} C'_k \geq pLatestTx_m + 1$. Então $U3 = \left\lfloor \sum_{j=1}^{n'} \frac{1}{q(j)} \right\rfloor$ é um limite superior válido se considerado o conjunto reduzido de mensagens.

O limite superior que se procura é determinado pelo mínimo dos quatro valores $U0, U1, U2, U3$: $U = \min(U0, U1, U2, U3)$. O resultado pode ser melhorado: dado um valor U para o limite superior, se $U > \left\lfloor \frac{\sum_{j=1}^{n'} C'_j - (3U - n')}{pLatestTx_m + 1} \right\rfloor$ então $U - 1$ é um limite superior válido.

Considerando-se o conjunto inicial de mensagens, o valor de $BusCycles(lf'(m,t))$ é dado por $BusCycles(lf'(m,t)) = R1 + R2 + R3 + U$.

A solução heurística para a definição do valor de w'_m parte da observação de que, em um cenário hipotético para o pior caso, uma mensagem m só será enviada no último instante possível no segmento dinâmico, o que resulta na Equação 3.18:

$$w'_m = ST_{bus} + pLatestTx_{N_p} \times gdMinislot \tag{3.18}$$

3.2.2.3 Considerações sobre as propostas em (Pop et al., 2008)

Apesar da reconhecida contribuição do trabalho apresentado em (POP et al., 2008) para o avanço do tema (HAGIESCU et al., 2007), existem considerações que devem ser feitas sobre o mesmo.

Em primeiro lugar, os próprios autores reconhecem que o alto custo computacional relacionado ao método exato proposto faz com que o uso do mesmo seja inviável para sistemas grandes.

Mas existe outro ponto importante a ser destacado em relação ao método exato. Durante a elaboração desta dissertação o mesmo foi implementado e sua execução mostrou que para certos conjuntos de mensagens o valor resultante para $BusCycles(lf(m,t))$ é otimista. Para ilustrar este problema, considere-se os dados apresentados na Tabela 3.1. Estes dados são utilizados em conjunto com a implementação do método exato apresentada no Apêndice A. Todos os valores estão

em MS.

Parâmetros	
gNumberOfMinislots	200
C **de** m **sob análise**	20
P **de** m **sob análise**	14500
D **de** m **sob análise**	14500
$pLatestTx_{N_p}$	180

Mensagem	**C**	**P**	**D**
m_1	80	14500	14500
m_2	80	14500	14500
m_3	80	14500	14500
m_4	80	14500	14500
m_5	20	14500	14500
m_6	20	14500	14500
m_7	20	14500	14500
m_8	20	14500	14500
m_9	20	14500	14500
m_10	20	14500	14500

Tabela 3.1: Parâmetros para exemplo com o método exato apresentado em (POP et al., 2008)

A computação do problema resulta em $BusCycles(lf(m,t)) = 0$, significando que mesmo no pior caso a mensagem m poderia ser transmitida no segmento dinâmico do primeiro ciclo. Entretanto, em pelo menos um dos possíveis cenários para a transmissão do conjunto de mensagens são necessários mais ciclos que o indicado como resultado. Por exemplo: em um dos possíveis cenários o segmento dinâmico do primeiro ciclo é ocupado por m_1, m_2, m_5 e 7 MS devido às demais mensagens não transmitidas, somando 187 MS utilizados. No segmento dinâmico do segundo ciclo são transmitidas m_3, m_4 e m_6, com 7 MS devido às demais mensagens, somando novamente 187 MS utilizados. Somente no terceiro ciclo a transmissão de m seria possível, de forma que, para este cenário, $BusCycles(lf(m,t)) = 2$, o que mostra claramente que o valor resultante do uso do método exato proposto é otimista.

O método exato foi implementado utilizando-se o GLPK (*GNU*

Linear Programming Kit) versão 4.29 (GLPK (GNU Linear Programming Kit), 2010), executado em um computador com processador Intel(R) Xeon(R) 2.00GHz, 3.2GB de memória RAM e Linux Debian 4.3.2 com *kernel* versão 2.6.29.

Existem também considerações a serem feitas em relação à heurística proposta para calcular $BusCycles(lf(mt))$. Segundo os autores, o pessimismo introduzido pela aproximação do tempo de computação de cada mensagem não é significante. No entanto, é fácil observar que, dependendo do número e do tamanho das mensagens envolvidas, o pessimismo introduzido é considerável. Por exemplo: considere-se um sistema com um conjunto de 400 mensagens no qual todas as mensagens possuem um tempo de comunicação igual a 2MS (um cenário realista, se considerados os veículos atuais e pacotes de dados para sensores e atuadores (NAVET; SIMONOT-LION, 2008)), sendo que a mensagem m_{400} sob análise é a de mais baixa prioridade, com $FrameID = 400$. O valor de C'_{400} (em MS), quando calculado com a Equação 3.17 será de:

$$C'_{400} = (400 - 1) + 2 = 401\text{MS}$$

um aumento de 200,5 vezes em relação ao tamanho original de m_{400}.

3.2.3 Artigo 3: "*Performance Analysis of FlexRay-based ECU Networks*"

O trabalho apresentado em (HAGIESCU et al., 2007) propõe o uso do *Real-Time Calculus* (RTC) para determinar diversos critérios de performance de sistemas baseados no FlexRay, como tempo de resposta de mensagens alocadas no segmento dinâmico de um barramento FlexRay, requerimentos de *buffer* e outros. Nesta seção será feita uma breve introdução ao RTC, sendo a seguir descrita a proposta apresentada no trabalho.

3.2.3.1 *Real-Time Calculus (RTC)*

O *Real-Time Calculus* (RTC) é um *framework* proposto em (CHAKRABORTY et al., 2003) que estende o *Network Calculus* (BOUDEC; THIRAN, 2001) para permitir a análise de sistemas de tempo real. Pode ser utilizado para derivar limites superiores e inferiores para vários critérios de performance como o máximo *delay* fim-a-fim de uma mensagem de dados ou requerimentos de *buffer*. Segundo (CHOKSHI; BHADURI, 2010), os conceitos chave do RTC são:

- São modelados padrões de chegada para tarefas ou mensagens que geram demandas para os recursos do sistema. Esta modelagem resulta no *modelo de eventos*.
- É modelado o serviço oferecido pelos recursos do sistema, resultando no *modelo de recursos*.
- É modelada a semântica do processamento.

O modelo de eventos é capturado pela noção de curvas de chegadas. Seja $R[s,t)$ o número de eventos que chegam em um intervalo de tempo $[s,t)$. Então, R, a curva de chegada superior $\bar{\alpha}^u$ e a curva de chegada inferior $\bar{\alpha}^l$ estão relacionados por $\bar{\alpha}^l(t-s) \leq R[s,t) \leq \bar{\alpha}^u(t-s) \forall s \leq t$ sendo $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$. Define-se a curva de chegada $\bar{\alpha} = [\bar{\alpha}^u, \bar{\alpha}^l]$. As curvas de chegada podem generalizar modelos tradicionais de eventos, como esporádicos, periódicos, periódicos com *jitter* ou qualquer outro padrão de chegada com comportamento temporal determinístico (WANDELER et al., 2006).

De modo similar ao modelo de eventos, o modelo de recursos é capturado pelas curvas de serviço. Seja $S[s,t)$ o número de eventos que um recurso pode fornecer no intervalo de tempo $[s,t)$. Então, S, a curva de serviço superior $\bar{\beta}^u$ e a curva de serviço inferior $\bar{\beta}^l$ estão relacionados por $\bar{\beta}^l(t-s) \leq S[s,t) \leq \bar{\beta}^u(t-s) \forall s \leq t$ sendo $\bar{\beta}^l(0) = \bar{\beta}^u(0) = 0$. Define-se a curva de serviço $\bar{\beta} = [\bar{\beta}^u, \bar{\beta}^l]$.

As curvas de chegada e de serviço também podem ser descritas em termos de quantidade de recursos, como por exemplo número de

ciclos de processador ou de comunicação, ao invés de número de eventos como descrito acima. Desta forma, a curva de serviços baseada em recursos $\beta(\Delta)$ representa as unidades do recurso disponível em qualquer intervalo de tempo de tamanho Δ, e a curva de chegadas baseada em recursos $\alpha(\Delta)$ representa as requisições em termos de unidades de recurso que chegam em qualquer intervalo de tempo Δ.

No RTC, o *Greedy Processing Component* (GPC) é um componente abstrato proposto em (CHAKRABORTY et al., 2003). É ativado toda vez que um evento descrito pela curva de chegada $\bar{\alpha}$ está disponível na entrada, e produz um único fluxo de saída descrito por $\bar{\alpha}^{'}$. Os eventos são processados de forma gulosa (*greedy*) em ordem FIFO (*First In, First Out*), sendo restringidos pela disponibilidade de recursos de processamento descritos pela curva de serviço β.

Seja E^u e E^l a representação da demanda de execução máxima e mínima em termos de unidades de recurso. Então, o GPC pode ser modelado como:

$$\bar{\alpha}^{'u} = \lceil \min\{(\bar{\alpha}^u \otimes \bar{\beta}^u) \oslash \bar{\beta}^l, \bar{\beta}^u\} \rceil \tag{3.19}$$

$$\bar{\alpha}^{'l} = \lfloor \min\{(\bar{\alpha}^l \oslash \bar{\beta}^u) \otimes \bar{\beta}^l, \bar{\beta}^l\} \rfloor \tag{3.20}$$

$$\bar{\beta}^{'u} = \max\{(\beta^u - \alpha^l) \bar{\oslash} 0, 0\} \tag{3.21}$$

$$\bar{\beta}^{'l} = (\beta^l - \alpha^u) \bar{\otimes} 0 \tag{3.22}$$

representando $\beta^{'}$ o serviço restante disponível para processar outros fluxos de eventos. O GPC é ilustrado graficamente na Figura 3.2(a). As definições para $\otimes$, $\oslash$, $\bar{\oslash}$ e $\bar{\otimes}$ podem ser encontradas em (BOUDEC; THIRAN, 2001).

O *wcrt* WR experimentado por qualquer evento no fluxo de eventos e o máximo número de eventos na fila de entrada $Buffer$ podem ser determinados como se segue:

$$WR \leq Del(\bar{\alpha}^u, \bar{\beta}^l) \tag{3.23}$$

$$Buffer \leq buf(\bar{\alpha}^u, \bar{\beta}^l) \tag{3.24}$$

em que $Del(\alpha^u, \beta^u) = \sup_{\Delta \geq 0}\{\inf\{\mu \geq 0 : \alpha^\mu(\Delta) \leq \beta^l(\Delta + \mu)\}\}$ e $Buf(\alpha^u, \beta^l) = \sup_{\lambda \geq 0}\{\alpha^u(\lambda) - \beta^l(\lambda)\}$. As definições para *sup* (ou *supremum*)e *inf* (ou *infimum*) podem ser encontradas em (BOUDEC; THIRAN, 2001).

Diversos componentes GPC podem ser interligados para formar uma rede de escalonamento (*scheduling network*), como por exemplo a rede de escalonamento para *rate monotonic* (RM) ilustrada na Figura 3.2(b) (HAGIESCU et al., 2007). O tempo de resposta fim-a-fim r experimentado por qualquer evento com uma curva de chegada superior $\bar{\alpha}^u$ processado em N GPCs consecutivos com curvas de serviço inferior $\bar{\beta}^l_l, ..., \bar{\beta}^l_N$ é:

$$r \leq Del(\bar{\alpha}^u, \bar{\beta}^l_1 \otimes ... \otimes \bar{\beta}^l_N). \tag{3.25}$$

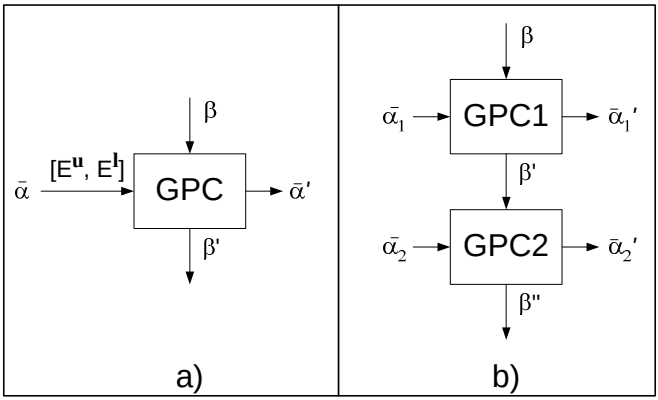

Figura 3.2: a) Greedy Processing Component (GPC) (CHOKSHI; BHADURI, 2010) b) Rede de escalonamento (HAGIESCU et al., 2007)

A utilidade do RTC pode ser verificada através de um exemplo. Considere duas tarefas T_1 e T_2 escalonadas por RM (Figura 3.3(a)). Ambas as tarefas são periódicas, sendo $P_1 = 4$ e $P_2 = 9$ unidades de tempo, $C_1 = 1$ e $C_2 = 2$ ciclos de processador. As curvas de chegada superior e inferior para a ativação de T_2 são ilustradas na Figura 3.3(b). Estas são similares à T_1, exceto pela diferença no tamanho do período.

As curvas do serviço oferecido por uma ECU sem carga são exibidas na Figura 3.3(c) (estas curvas coincidem por razões óbvias).

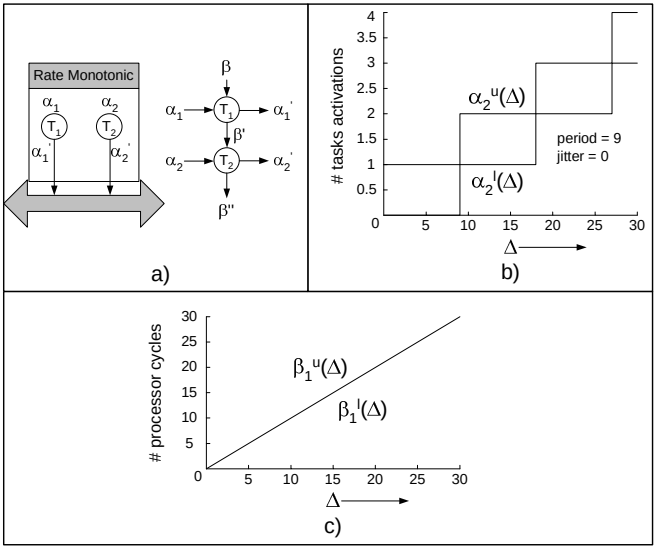

Figura 3.3: a) Escalonamento *rate-monotonic* para duas tarefas e rede de escalonamento correspondente. b) Curvas α^u e α^l correspondentes a uma ativação periódica. c) Curvas β^u e β^l para um processador sem carga. (HAGIESCU et al., 2007)

Utilizando-se α_1^u, α_1^l e β_1^u, β_1^l pode-se computar os valores para $\beta_1^{u'}$, $\beta_1^{l'}$ que são os limites do serviço que estará disponível para a tarefa T_2. As curvas $\beta_1^{u'}$, $\beta_1^{l'}$ são ilustradas na Figura 3.4(a), e a curva de chegada α_2 para as mensagens geradas pela tarefa T_2 é exibida na Figura 3.4(b).

Maiores informações sobre *Network Calculus* e *Real-Time Calculus* podem ser encontradas em (BOUDEC; THIRAN, 2001), (CHAKRABORTY et al., 2003) e (WANDELER et al., 2006).

3.2.3.2 Modelo de Sistema

O trabalho apresentado em (HAGIESCU et al., 2007) considera arquiteturas que consistem de múltiplas Unidades Eletrônicas de Con-

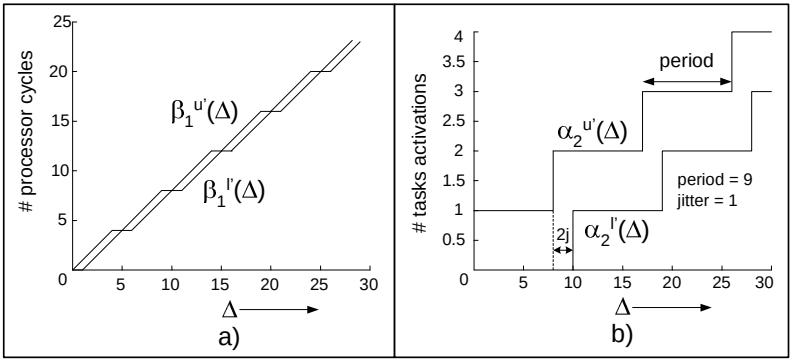

Figura 3.4: a) Limites superior e inferior para o serviço restante após o processamento de uma tarefa T_1. b) Limites para as mensagens geradas pela tarefa T_2 (HAGIESCU et al., 2007)

trole (*Electronic Control Units*, ECU) ligadas a um barramento FlexRay. Uma ou mais aplicações são particionadas em tarefas que por sua vez são mapeadas em diferentes ECUs. Uma ECU que executa múltiplas tarefas utiliza um escalonador para compartilhar os recursos de processamento existentes, como mostrado na Figura 3.5. Cada tarefa ou é ativada a uma determinada taxa ou é disparada por um evento de saída de outra tarefa. Uma vez ativada, a tarefa deve ser processada e consome um número fixo de ciclos de processador da ECU na qual está sendo executada. Também é considerado que as mensagens de dados são alocadas no segmento dinâmico do barramento. O tipo de mensagens com as quais a proposta trabalha não é definido de forma explícita no artigo, mas com base no texto pode-se deduzir que as mensagens analisadas podem ser esporádicas.

3.2.3.3 Método Proposto

Segundo os autores da proposta, as maneiras normalmente utilizadas para modelar no RTC os limites de um serviço oferecido β não podem ser aplicadas para o FlexRay devido às restrições deste protocolo. Por exemplo, uma maneira de modelar o sistema seria como um

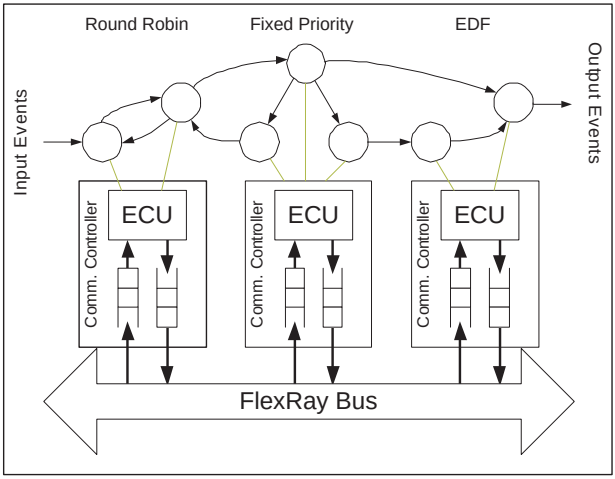

Figura 3.5: Modelo de Sistema (HAGIESCU et al., 2007)

escalonador de prioridade fixa similar ao descrito na Seção 3.2.3.1, onde β indica o serviço oferecido e sucessivos $\beta^{'}s$ são computados a partir dos tamanhos das mensagens do sistema. Entretanto, essa abordagem não funciona devido às seguintes propriedades do FlexRay: (i) uma tarefa só pode enviar uma mensagem se esta couber na fração restante do DYN. (ii) Se uma tarefa perder seu *slot* no ciclo atual, deve esperar o ciclo seguinte para enviar uma mensagem. (iii) Uma tarefa pode enviar no máximo uma mensagem por segmento dinâmico e (iv) um *minislot* é consumido toda vez que uma tarefa não está pronta para transmitir uma mensagem no início de seu *slot*.

Para comprovar que não é possível modelar o FlexRay como um escalonador de prioridade fixa é apresentado um exemplo que utiliza a Figura 3.6, na qual é ilustrada a curva de chegada α^u de uma mensagem com tamanho de 10 MS que deve ser transmitida em um barramento FlexRay cujo segmento dinâmico possui tamanho de 8MS. Assume-se que o tamanho de um ciclo de comunicação é igual a p unidades de tempo, o DYN possui tamanho igual a d unidades de tempo e um MS tem tamanho de uma unidade de tempo. Nota-se que durante um in-

tervalo de tempo $\Delta \leq (p-d)$ não existe disponibilidade de serviço para o segmento dinâmico devido ao efeito bloqueante do segmento estático. Como o tamanho da mensagem é maior que o tamanho do segmento dinâmico, esta nunca será transmitida. Entretanto, se o sistema for modelado como um escalonador de prioridade fixa, o *framework* modela a mensagem como sendo transmitida em dois ciclos de comunicação, incorrendo em um atraso igual a máxima distância horizontal entre α^u e β^l (Figura 3.6).

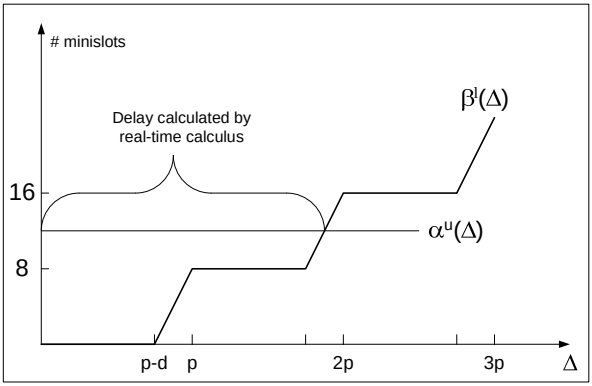

Figura 3.6: Máximo atraso de α^u e β^l (HAGIESCU et al., 2007)

No exemplo acima é demonstrado que as curvas de serviço $\beta^u(\Delta)$ e $\beta^l(\Delta)$ capturam o serviço total disponível no segmento dinâmico, mas é necessário modelar quanto deste serviço pode realmente ser utilizado por mensagens dinâmicas. A proposta apresentada em (HAGIESCU et al., 2007) para esta modelagem é descrita a seguir.

Supõe-se que $T_1, ..., T_n$ são tarefas que enviam mensagens no segmento dinâmico de um barramento FlexRay, com cada mensagem enviada pela tarefa T_i sendo denotada como m_i e possuindo um tamanho k_i (em MS). O tamanho do segmento dinâmico é de k MS (ou d unidades de tempo), o tamanho de um ciclo de comunicação é de p unidades de tempo e cada minislot tem tamanho de 1MS. Assume-se que $\beta = [\beta^u, \beta^l]$ é a curva do serviço oferecido (em MS) por um segmento dinâmico sem carga (ou seja, o serviço total para todas as mensagens

dinâmicas). O que se procura obter é $\beta_i = [\beta_i^u, \beta_i^l]$, que é o serviço realmente oferecido para uma mensagem m_i.

De acordo com (HAGIESCU et al., 2007), para que a curva β_i^l (ou seja, o serviço oferecido no pior caso para a mensagem dinâmica de prioridade mais alta) possa ser obtida, β^l deve ser transformada de acordo com os passos descritos a seguir:

1. Extrair k_1 MS de β^l a cada ciclo de comunicação. Invalidar os ciclos de comunicação que contém menos de k_1 MS. Isto visa modelar que, em qualquer ciclo de comunicação, no máximo k_1 MS estão disponíveis para m_1 (Figura 3.7(a)).

2. Discretizar o limite de serviço obtido do passo 1. Isto é feito pelo fato de que uma mensagem não pode utilizar mais que um ciclo de comunicação (Figura 3.7(a)).

3. Deslocar em d unidades de tempo a curva de serviço obtida no passo 2. Isto é feito para modelar o fato de que se uma mensagem ficar pronta imediatamente após o início do seu *slot*, ela deverá esperar o ciclo de comunicação seguinte (Figura 3.7(b)).

4. Um MS é perdido sempre que uma tarefa não transmite mensagens no seu *slot*. Isto é considerado pela subtração de um MS de cada ciclo de comunicação corroborado com um ajuste no tamanho da mensagem m_1 nas análises subsequentes do serviço consumido por mensagens da tarefa T_1.

A curva de serviço resultante β_i^l captura o serviço mínimo (ou garantido) fornecido pelo segmento dinâmico para uso de mensagens da tarefa T_1. Desta forma, β_1^l pode ser utilizada com o GPC descrito na Seção 3.2.3.1 para computar as propriedades temporais de qualquer mensagem m_1.

O serviço disponível para tarefas de prioridade mais baixa é composto por dois componentes: (i) o serviço restante após a transformação 1 dado pela curva inferior $\bar{\beta}^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta}\{\beta^l(\lambda) - \beta_1^l(\lambda)\}$ e (ii) o serviço que não foi utilizado por T_1, computado a partir de β_1^l e α_1^u

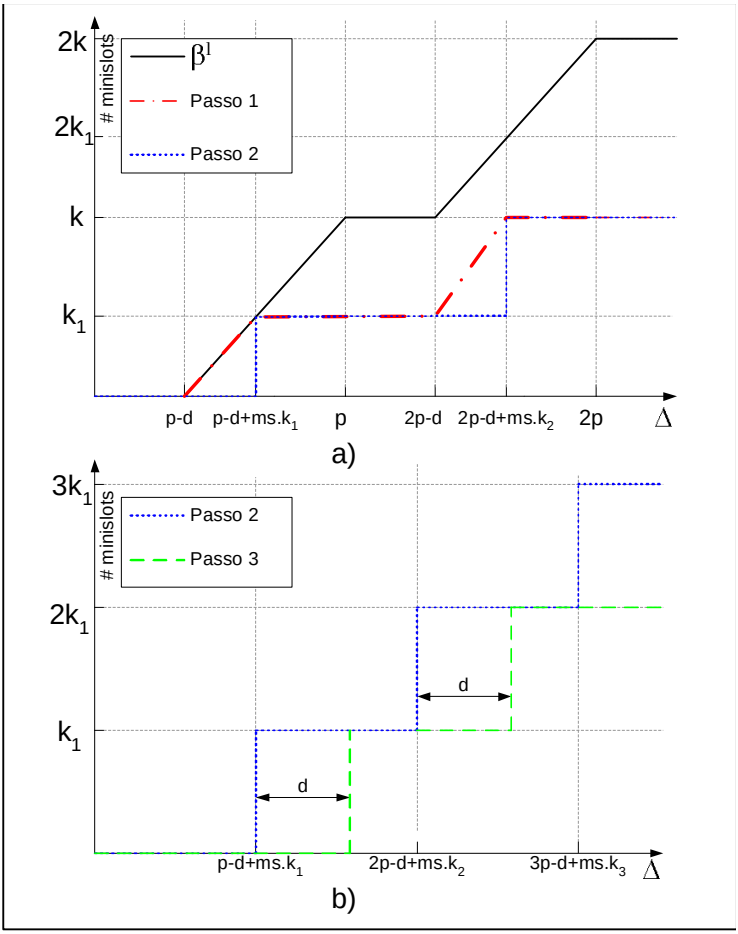

Figura 3.7: (a) Passos 1 e 2 e (b) Passo 3 para obter a transformação de β_1^l (HAGIESCU et al., 2007)

e capturado pela curva $\beta_1^{l'}$. No entanto, $\beta_1^{l'}$ não pode ser adicionada diretamente a $\bar{\beta}^l$ por ser relativa a mensagens geradas pela tarefa T_1, sendo primeiro ser necessária sua transformação aplicando o "'inverso"' dos passos 2 e 3 descritos anteriormente. A soma destas duas funções representa o serviço disponível para tarefas de prioridade mais baixa, e deve ser transformado da mesma maneira que β^l, mas utilizando as

informações específicas para mensagens da tarefa T_2. Este procedimento é repetido para todas as tarefas $T_3, ..., T_n$. Uma síntese deste procedimento é ilustrado na Figura 3.8.

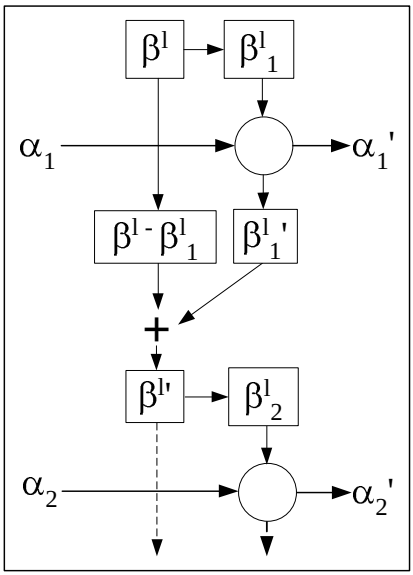

Figura 3.8: Visão do procedimento proposto em (HAGIESCU et al., 2007)

3.2.3.4 Considerações sobre a proposta em (Hagiescu et al., 2007)

A proposta apresentada em (HAGIESCU et al., 2007) é diretamente questionada em (CHOKSHI; BHADURI, 2010), e por este motivo esta não foi implementada durante o desenvolvimento desta dissertação. Os motivos que levam ao questionamento serão reproduzidos na próxima seção.

3.2.4 Artigo 4: "*Performance Analysis of FlexRay-based Systems using Real-Time Calculus, Revisited*"

O trabalho apresentado em (CHOKSHI; BHADURI, 2010) questiona o método proposto em (HAGIESCU et al., 2007), mostrando que a análise anterior pode levar a resultados otimistas e, em específico, mostra que obter a curva de serviço superior não é trivial, e não segue o mesmo raciocínio empregado para a curva de serviço inferior. Também oferece um novo modelo que utiliza o RTC e que adicionalmente permite a análise de mensagens de tamanho variável no segmento dinâmico do FlexRay. Por ser um questionamento sobre o outro método, o trabalho em (CHOKSHI; BHADURI, 2010) assume os mesmos modelos de sistema e mensagens que os assumidos na proposta original em (HAGIESCU et al., 2007) (ou seja, sistemas em que é possível a transmissão de mensagens periódicas e esporádicas).

3.2.4.1 Questionamentos sobre o limite de serviço superior disponível para uma mensagem

Segundo (CHOKSHI; BHADURI, 2010), o trabalho em (HAGIESCU et al., 2007) não aborda a questão de como definir a curva de serviço superior β^u que está disponível para uma mensagem. Como foi brevemente discutido na Seção 3.2.3.1, o GPC necessita tanto da curva de serviço superior β^u como da curva de serviço inferior β^l para que possa ser utilizado.

Uma possibilidade para calcular β^u seria com o mesmo método proposto em (HAGIESCU et al., 2007), sem no entanto executar o passo 3. Mas em (CHOKSHI; BHADURI, 2010) é demonstrado que calcular a curva superior com qualquer um dos dois métodos abaixo levará resultados errôneos:

- Método 1: utilizando os passos 1 e 2 propostos em (HAGIESCU et al., 2007)

- Método 2: utilizando apenas o passo 1 proposto em (HAGIESCU et al., 2007)

Para a demonstração, é considerado um sistema com duas ECU's (ambas utilizando FPS) e um barramento FlexRay, como ilustrado na Figura 3.9, na qual a tarefa T_x (prioridade mais alta) da ECU_1 gera a mensagem m_1 que está alocada no segmento dinâmico. A tarefa T_y está alocada na ECU_2 e é ativada pela recepção de m_1. A tarefa T_z na ECU_2 é ativada após a finalização de T_y e envia a mensagem m_2 para a tarefa T_v. Cada MS tem tamanho de uma unidade de tempo. O ciclo FlexRay tem tamanho de 10 MS, sendo que destes 7 MS correspondem ao segmento dinâmico. A tarefa T_x é ativada periodicamente e tem $P = 21MS$ e $C = 3MS$. A tarefa T_y tem $C = 2MS$, e m_1 tem $C = 4MS$.

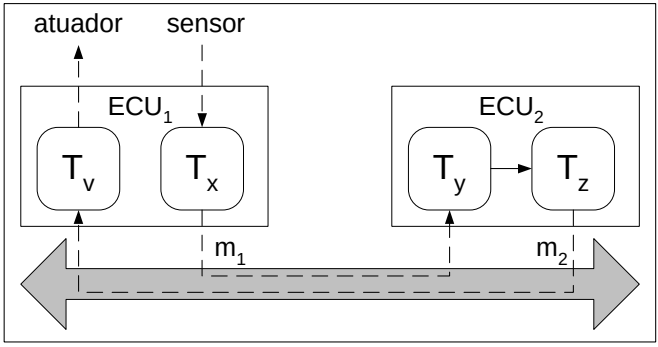

Figura 3.9: Exemplo de sistema (CHOKSHI; BHADURI, 2010)

Limite superior de serviço utilizando-se o Método 1: Seja a curva de chegada de m_1 $\bar{\alpha}_1 = [\bar{\alpha}_1^u, \bar{\alpha}_1^l]$. Na Figura 3.10(a) são ilustradas as curvas de serviço β_1 disponíveis para m_1 (β_1^u é calculada com o Método 1). A curva de saída $\bar{\alpha}_1^{'}$ (ilustrada na Figura 3.10(b)) é computada pelas Equações 3.19 e 3.20 utilizando-se $\bar{\alpha}_1$ e β_1. Pode-se notar pela Figura 3.10(b) que $\bar{\alpha}_1^u$ é incorreta, pois a mesma indica que em qualquer intervalo $\Delta < 4MS$ existem no máximo zero mensagens no fluxo de saída, o que é incorreto, pois uma mensagem pode aparecer

na saída em qualquer instante. Pelo método proposto, $\bar{\alpha}_1'^{u}$ é utilizado como curva de chegada para a tarefa T_y, sendo que a curva de serviço inferior restante após o processamento de T_y é exibida na Figura 3.10(c). Pode-se verificar que esta também é incorreta, pois indica que em qualquer intervalo de 2 unidades de tempo o mínimo serviço disponível é de 2 ciclos, o que não é o caso, pois em qualquer intervalo de 2 unidades de tempo o mínimo serviço disponível é 0, dado que a tarefa T_y (de maior prioridade) pode estar ocupando a ECU.

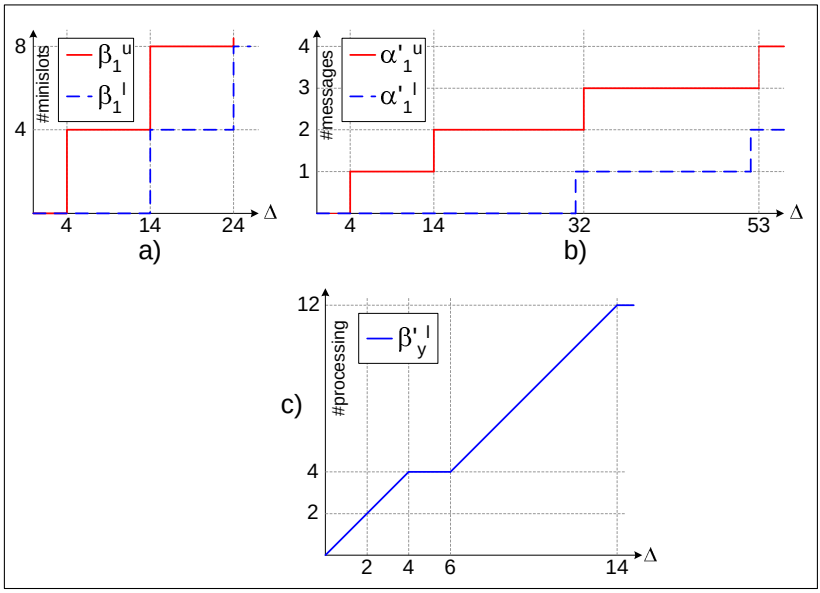

Figura 3.10: a) β_1 para o exemplo. b) $\bar{\alpha}_1'$ para o exemplo. c) $\beta_y'^{l}$ para o exemplo. (CHOKSHI; BHADURI, 2010)

Limite superior de serviço utilizando-se o Método 2: Se for utilizado o Método 2 para a obtenção da curva de serviço superior β_1^u, então o limite superior do serviço inutilizado por m_1 é incorreto, como pode ser observado na Figura 3.11. Em uma situação real, em qualquer ciclo ou todo o serviço disponível para uma mensagem é deixado sem uso ou é consumido imediatamente. No entanto, na Figura 3.11 pode-se observar que no ciclo de comunicação entre o intervalo 10

e 20, apenas um MS é deixado sem uso, o que é incorreto. Além disso, uma curva de serviço incorreta levará a uma análise otimista quando for utilizada para se obter o serviço disponível para m_2.

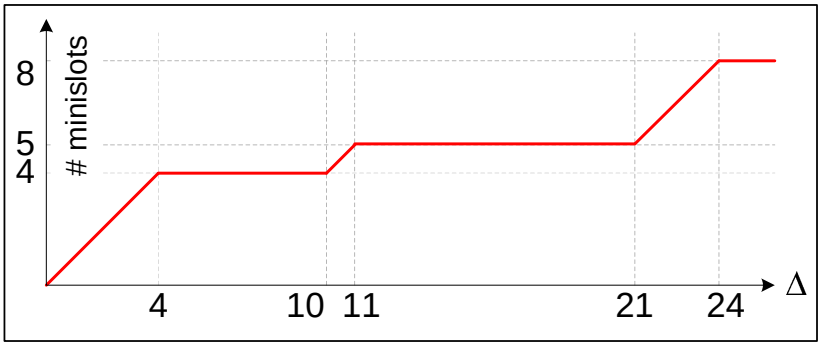

Figura 3.11: Limite superior para o serviço não utilizado por m_1 (CHOKSHI; BHADURI, 2010)

3.2.4.2 Método Proposto

O método proposto em (CHOKSHI; BHADURI, 2010) assume que, diferente do método em (HAGIESCU et al., 2007) no qual são assumidas mensagens de tamanho fixo, uma mensagem m_i possui um mínimo de k_i^l e um máximo de k_i^u MS.

Uma mensagem dinâmica do FlexRay é modelada como ilustrado na Figura 3.12. Este modelo é baseado no fato de que o tempo de resposta de uma mensagem m consiste de um tempo de espera e um tempo de comunicação. O tempo de espera é o tempo desde a chegada até o momento em que m consegue acesso ao barramento para iniciar a transmissão. O tempo de comunicação é o tempo realmente utilizado para a transmissão de m (C_m/bus_speed). Portanto, existem dois GPCs (GPC-W e GPC-C) no modelo da mensagem.

Para que o GPC-W represente apenas o tempo de espera de m_i é necessário modificar o serviço total disponível para m_i ($\beta_i^t = [\beta_i^{tu}, \beta_i^{tl}]$) de modo a obter a curva de serviço $\beta_i^t = [\beta_i^{tu}, \beta_i^{tl}]$. O tempo de comunicação de m_i será considerado pelo GPC-C, cuja curva de serviço

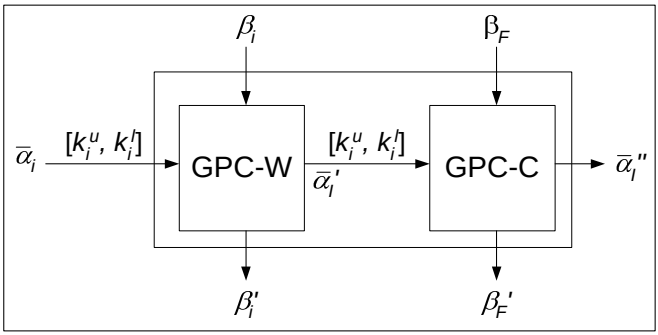

Figura 3.12: Modelo RTC para a mensagem m_i (CHOKSHI; BHADURI, 2010)

de entrada β_F é o "'serviço total"' com capacidade igual à largura de banda do barramento FlexRay. Como a mensagem m_1 possui a mais alta prioridade no sistema é considerado para a mesma $\beta_1^t = \beta$, onde β é o serviço total oferecido por um segmento dinâmico sem carga.

Para se obter β_i^l, é aplicado o seguinte procedimento:

1. Tornar nulos os ciclos de comunicação de β_i^{tl} que contém menos que k_i^u MS. Extrair k_i^u MS dos ciclos de comunicação restantes. A curva resultante é $\beta_{i,1}^{tl}$ (Figura 3.13(a)).

2. Para cada incremento de segmento na curva $\beta_{i,1}^{tl}$, discretizar o segmento no ponto em que a mesma inicia o incremento. A curva resultante é $\beta_{i,2}^{tl}$ (Figura 3.13(a)).

3. Deslocar a curva $\beta_{i,2}^{tl}$ d unidades de tempo para obter β_i^l (Figura 3.13(b)).

Para obter β_i^u é aplicado o seguinte procedimento:

1. Tornar nulos os ciclos de comunicação de β_i^{tu} que contém menos que k_i^l MS. Extrair k_i^l MS dos ciclos de comunicação restantes. A curva resultante é $\beta_{i,1}^{tu}$ (Figura 3.14).

2. Para cada incremento de segmento na curva $\beta_{i,1}^{tu}$, discretizar o segmento no ponto em que começa o incremento do mesmo. A

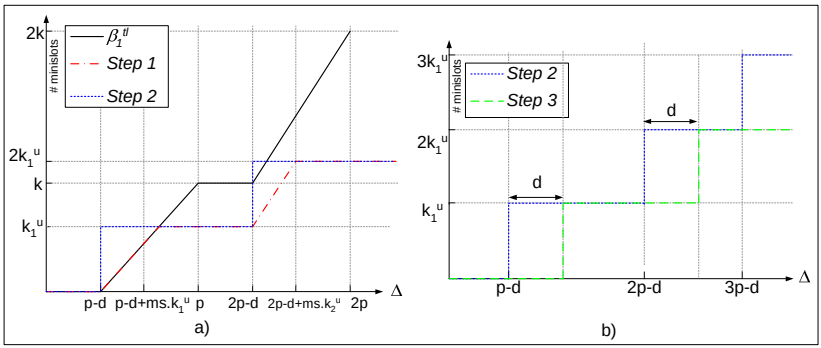

Figura 3.13: a) Passos 1 e 2 e b) Passo 3 para obter β_1^l (CHOKSHI; BHADURI, 2010)

curva resultante é β_i^u (Figura 3.14).

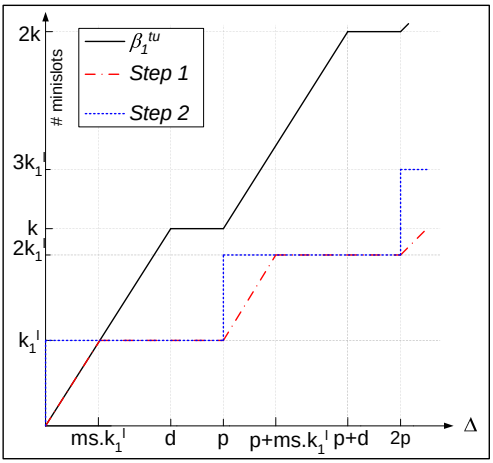

Figura 3.14: Passos 1 e 2 para obter β_1^u (CHOKSHI; BHADURI, 2010)

O wcrt da mensagem é calculado através da Equação 3.25 com as curvas de serviço β_i^l e β_F^l e a curva de chegadas α_i.

O serviço não utilizado por m_i, $\beta_i^{'} = [\beta_i^{'u}, \beta_i^{'l}]$ é obtido com o uso das Equações 3.21 e 3.22. Mas este serviço é específico à mensagem m_i, portanto deve ser aplicado em $\beta_i^{'l}$ o inverso dos Passo 2 e 3 com o

objetivo de se obter $\beta'^{l}_{i,x}$.

Para que a propriedade do *minislot* não utilizado seja considerada, subtrai-se um MS de cada ciclo de comunicação de $\beta'_{i,x}$ que ofereça mais que 0MS de serviço com o objetivo de se obter $\beta'_{i,y}$.

O serviço indisponível para m_i é $\beta^n_i = [\beta^{nu}_i, \beta^{nl}_i]$, onde $\beta^{nu}_i(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta^{tu}_i(\lambda) - \beta^{tu}_{i,1}(\lambda)\}$ e $\beta^{nl}_i(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta^{tl}_i(\lambda) - \beta^{tl}_{i,1}(\lambda)\}$.

Com isso, o serviço oferecido para a próxima mensagem de maior prioridade m_{i+1} é $\beta^t_{i+1} = \beta^n_i + \beta'_{i,y}$.

3.2.4.3 Considerações sobre a proposta em (Chokshi and Bhaduri, 2010)

O método apresentado em (CHOKSHI; BHADURI, 2010) foi implementado para que pudesse ser comparado com outras soluções. No entanto, para certos conjuntos de mensagens e parâmetros este método apresentou resultados otimistas. Para ilustrar este problema, considere-se como exemplo um sistema FlexRay com os parâmetros da Tabela 3.2. Todos os valores são dados em MS, e as mensagens são esporádicas.

Parâmetros	
ST_{bus}	10
gNumberOfMinislots	10

Mensagem	**C**	**MIT**
m_1	3	1000
m_2	3	38
m_3	3	55
m_4	4	1000

Tabela 3.2: Parâmetros para exemplo com o método exato apresentado em (CHOKSHI; BHADURI, 2010)

A implementação do método proposto apresenta como tempo de resposta para cada mensagem os valores ilustrados na Tabela 3.3. No entanto, como pode ser observado na Figura 3.15, o pior cenário de transmissão para m_4 resulta em um tempo de resposta de 63MS, e não 48MS como o método proposto indica.

Mensagem	Tempo de Resposta
m_1	23
m_2	26
m_3	29
m_4	48

Tabela 3.3: Tempos de resposta com o método proposto em (CHOKSHI; BHADURI, 2010) para o exemplo da Tabela 3.2

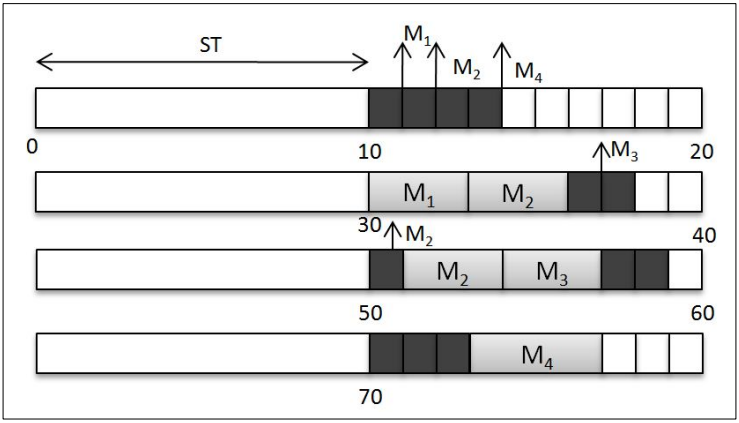

Figura 3.15: Pior caso de transmissão de mensagens para o sistema apresentado na Tabela 3.2

O método em (CHOKSHI; BHADURI, 2010) foi implementado utilizando-se o RTC Toolbox Version 1.2.beta.50 (WANDELER; THIELE, 2006), executado em um computador com processador Intel(R) Xeon(R) 2.00GHz, 3.2GB de memória RAM e Microsoft Windows XP SP3. A implementação utilizada pode ser encontrada no Anexo B deste documento.

3.3 Conclusões

Neste capítulo foram discutidos trabalhos relacionados ao FlexRay encontrados na literatura. Alguns destes trabalhos apresentam métodos para a definição de parâmetros para sistemas FlexRay (Artigo 1), enquanto outros abordam a análise do tempo de resposta de

mensagens FlexRay (Artigos 2, 3 e 4).

Como o foco desta dissertação é a análise do tempo de resposta de mensagens alocadas no segmento dinâmico, neste capítulo foi dada ênfase à discussão de trabalhos relacionados a este assunto. Os quatro artigos discutidos são listados na Tabela 3.4, que apresenta o nome do artigo, o tipo de abordagem (se é um método heurístico, exato ou outro), as classes de mensagens que podem ser analisadas (periódicas, esporádicas ou ambas) e comentários sobre as propostas.

A principal conclusão que pode ser tirada da observação da Tabela 3.4 é que, das propostas atualmente disponíveis na literatura, o único método que poderia ser utilizado de forma confiável para projetos de sistemas FlexRay com mensagens alocadas no segmento dinâmico é o método heurístico proposto em (POP et al., 2008), sendo que mesmo esta solução possui limitações no seu uso devido ao pessimismo acrescentado.

No próximo capítulo serão apresentadas as propostas de métodos alternativos para análise de propriedades temporais de mensagens alocadas no segmento dinâmico do FlexRay que foram desenvolvidas durante a elaboração desta dissertação de Mestrado.

Trabalho	Abordagem	Tipos de mensagem	Comentários
Artigo 1: (Schmidt and Schmidt, 2008b)	Utiliza ILP para obter parâmetros ótimos para um sistema FlexRay	Periódicas e esporádicas	O método não tem seu estudo aprofundado por não apresentar análise de tempo de resposta
Artigo 2: (Pop et al., 2008), método exato	Utiliza ILP para obter o wcrt exato de mensagens dinâmicas	Periódicas e esporádicas	O método pode levar a resultados otimistas, e o tempo de computação é elevado
Artigo 2: (Pop et al., 2008), método heurístico	Método heurístico para obter um limite superior para o wcrt das mensagens	Periódicas e esporádicas	O resultado final pode ser altamente pessimista
Artigo 3: (Hagiescu et al., 2007)	Curvas de serviço do RTC	Periódicas e esporádicas	O método proposto é contestado em outro trabalho
Artigo 4: (Chokshi and Bhaduri, 2010)	Curvas de serviço do RTC	Periódicas e esporádicas	Contesta o trabalho em (Hagiescu et al, 2007), apresentando uma nova solução. No entanto, pode indicar tempo de resposta otimista

Tabela 3.4: Resumo sobre os artigos relacionados à análise do tempo de resposta no DYN

Capítulo 4

Contribuições para a Análise de Escalonabilidade no FlexRay

No Capítulo 3 foram apresentados trabalhos da literatura que abordam a análise de escalonabilidade do segmento dinâmico do FlexRay. Mas, como discutido naquele capítulo, as soluções apresentadas podem levar a resultados otimistas ou, no caso do método heurístico, a um resultado bastante pessimista para o tempo de resposta das mensagens sob análise.

Neste capítulo é inicialmente apresentada uma discussão sobre a dificuldade em se modelar o FlexRay para fins de análise (Seção 4.1) sendo a seguir apresentadas quatro propostas de métodos para a análise de escalonabilidade de mensagens alocadas no segmento dinâmico do FlexRay. A primeira proposta (Seção 4.2) utiliza um modelo em Redes de Petri para a análise. As outras três propostas são apresentadas nas seções 4.3, 4.4 e 4.5, sendo métodos heurísticos para a computação do

tempo de resposta no pior caso de mensagens esporádicas alocadas no segmento dinâmico do FlexRay.

4.1 Problemas em Modelar o Segmento Dinâmico do FlexRay Para Fins de Análise

Como discutido no Capítulo 2 deste documento, o segmento dinâmico do FlexRay apresenta o seguinte comportamento:

1. Se um nó é autorizado a transmitir, existem dados prontos para serem enviados e existem MS em número suficiente para a transmissão, o *slot* associado à mensagem utilizará o número necessário de MS para a transmissão dos dados.

2. Se um nó é autorizado a transmitir, existem dados prontos para serem enviados mas não existem MS disponíveis para a transmissão do quadro completo, o *slot* associado à mensagem utiliza apenas um MS e a transmissão dos dados é postergada para o ciclo seguinte.

3. Não existem dados prontos para serem enviados em um determinado *slot*. Neste caso o *slot* utiliza apenas um MS.

4. O contador de MS fica vazio ($MSCounter = 0$) ou $vSlotCounter$ atinge $cSlotIDMax$. Neste caso o segmento dinâmico é finalizado e é iniciado um novo ciclo FlexRay.

Qualquer análise para o tempo de resposta de mensagens alocadas no segmento dinâmico deve levar em consideração este comportamento.

Se o segmento dinâmico for composto apenas por mensagens periódicas é possível utilizar as definições do comportamento para construir uma linha de tempo que indica o tempo de resposta no pior caso para uma mensagem m alocada neste segmento. No entanto, se houverem mensagens esporádicas alocadas no segmento dinâmico, definir

o pior cenário para a transmissão de uma mensagem m_i sob análise torna-se um problema combinatório. Como este problema é similar ao problema da mochila [1], fica claro que o mesmo é np-completo (BLACK, 2010b). Assim sendo, mesmo sendo possível desenvolver um método para a definição do valor exato, o custo computacional deste método possivelmente o tornaria inviável para uso prático.

Por este motivo, os métodos para a análise de sistemas com mensagens esporádicas propostos neste capítulo utilizam heurísticas para encontrar um limite superior no tempo de resposta das mensagens. No entanto, mesmo supor premissas para estes métodos é complicado, como é ilustrado a seguir.

Sabe-se que a quantidade mínima de *minislots* que devem ser ocupados em um segmento dinâmico para evitar a transmissão de uma mensagem dinâmica m_i em um ciclo equivale a $pLatestTx_{m_i} + 1$ MS[2]. Dado este fato, um sistema FlexRay onde existem mensagens esporádicas alocadas no segmento dinâmico, uma mensagem dinâmica m_i para a qual se deseja obter o tempo de resposta no pior caso e um conjunto $hp(m_i)$ contendo as mensagens dinâmicas com prioridade mais alta que m_i, seria possível desenvolver um método para obter um limite superior do tempo de resposta de m_i no qual o conjunto $hp(m_i)$ é reduzido a outro conjunto $hp'(m_i)$ pela seguinte técnica: quaisquer duas mensagens m_j e $m_k \in hp(i)$ cuja soma dos tempos de transmissão seja igual a $pLatestTx_{mi} + 1$ MS podem ser removidas de $hp(m_i)$ em troca da adição de um ciclo FlexRay completo no tempo de resposta de m_i. A posterior análise de $hp'(m_i)$ por alguma outra técnica daria um limite superior para o tempo de resposta de m_i. A técnica para redução descrita acima utiliza a seguinte premissa: como basta a utilização de $pLatestTx_{m_i} + 1$ MS para evitar a transmissão de m_i no ciclo corrente, uma combinação de duas mensagens que ocupe mais que este valor se-

[1] O problema da mochila é um problema clássico utilizado em otimização e tem a seguinte definição: existe uma mochila com capacidade $c > 0$ e N itens. Cada item tem valor $v_i > 0$ e peso $w_i > 0$. Deve-se encontrar uma seleção de itens que caibam na sacola maximizando a relação entre o custo e o peso (BLACK, 2010a)

[2] O valor $pLatestTx$ é explicado no Capítulo 2, e equivale ao número do último *slot* dinâmico no qual é possível a transmissão de uma determinada mensagem.

ria "desperdício", pois utilizar apenas o valor mínimo significaria que mais *minislots* seriam necessários em ciclos futuros.

Entretanto, a premissa da técnica descrita acima não é válida, como é demonstrado em um sistema FlexRay que utiliza os parâmetros da Tabela 4.1. Para este sistema, a mensagem que se deseja analisar é m_5, e $hp(m_5)$ contém as mensagens m_1, m_2, m_3 e m_4. Se for utilizada a técnica de redução descrita no parágrafo anterior, m_1 e m_2 seriam retiradas do conjunto pois $C_1 + C_2 = 16MS = pLatestTx_{m_5} + 1$ equivalendo um ciclo FlexRay completo. Para o conjunto $hp'(m_5)$ resultante seriam possíveis as seguintes combinações: i) são utilizados 2MS equivalentes a m_1 e m_2, m_3 é transmitida, não existe folga para a transmissão de m_4 e m_5 é transmitida com um tempo de resposta para $m_5 = gdCycle + 1 + 1 + C_3 + 1 + C_5 = 47$MS. ii) são utilizados 2MS equivalentes a m_1 e m_2, m_3 não está pronta e portanto é gasto 1MS e m_4 é transmitida com um tempo de resposta para $m_5 = gdCycle + 1 + 1 + 1 + C_4 + C_5 = 47$MS. Mas existe uma situação pior que não é capturada pela técnica acima: no primeiro ciclo poderiam ser transmitidas m_1 e m_3, no segundo ciclo m_2 e m_4 e somente no terceiro ciclo seria possível a transmissão de m_5, o que daria um tempo de resposta para $m4 = 2 * gdCycle + C_5 1 + 1 + 1 + 1 = 69$MS.

$gdCycle$	30MS
ST_{bus}	11MS
$gNumberOfMinislots$	19MS
Mensagem C **(MS)**	$D = P$ **(MS)**
m_1 8	100
m_2 8	100
m_3 9	100
m_4 9	100
m_5 5	100
$pLatestTx_{m_5}$ 15	

Tabela 4.1: Exemplo de valores para o problema combinatório

No exemplo acima, fica claro o problema combinatório relacionado à análise de mensagens esporádicas alocadas no segmento dinâ-

mico do FlexRay, bem como a dificuldade de se supor premissas para uso em métodos heurísticos.

4.2 Análise do DYN Com Redes de Petri

A primeira proposta apresentada neste documento é um método que utiliza Redes de Petri e técnicas de *model checking* para analisar a escalonabilidade de mensagens alocadas no segmento dinâmico de um sistema que utiliza FlexRay.

No início desta seção são apresentados de forma breve conceitos sobre *model checking* para, a seguir, descrever o método proposto. No final da seção são feitas considerações sobre esta proposta.

4.2.1 *Model Checking*

De acordo com (CLARKE et al., 1999), o *model checking* é uma técnica para verificação de sistemas reativos finitos, como protocolos de comunicação ou projetos de circuitos. As especificações são expressas em lógica temporal, e o sistema reativo é modelado como um grafo de estados. Um procedimento eficiente de busca é utilizado para determinar se o grafo de estados satisfaz as especificações.

As metodologias para *model checking* usualmente requerem três elementos: uma descrição formal do sistema, uma linguagem de especificação onde possam ser expressas as propriedades do sistema que se deseja investigar e um procedimento de decisão que permita checar a conformidade entre a descrição do sistema e sua especificação (NASPOLINI, 2006).

No restante desta seção será descrita de forma breve uma linguagem para a modelagem formal de sistemas conhecida como *Time Petri Net* (TPN), serão abordadas linguagens para lógica temporal e também apresentada uma ferramenta para *model checking* chamada Roméo.

4.2.1.1 Time Petri Net

Uma Rede de Petri (*Petri Net*, PN) é um modelo matemático definido por Carl A. Petri que pode ser utilizado para a descrição de sistemas concorrentes. Uma PN é definida como uma tupla $R =< Pl, Tr, B, F >$ na qual $Pl = \{p_1, p_2, ...p_n\}$ é o conjunto de lugares, $Tr = \{t_1, t_2, ...t_n\}$ é o conjunto de transições, $B = Tr \times Pl \rightarrow \mathbb{N}$ é a função de incidência anterior e $F = Tr \times Pl \rightarrow \mathbb{N}$ é a função de incidência posterior (BERTHOMIEU; DIAZ, 1991). Uma PN pode ser representada graficamente, com retângulos representando as transições e círculos representando os lugares. Em uma rede, uma transição tr_i é dita "'habilitada"' se cada lugar de entrada p possui ao menos $w(p, tr_i)$ fichas (também chamadas de *tokens*), sendo que $w(p, tr_i)$ é o peso do arco de p para tr_i. Uma transição pode ser disparada após estar habilitada, e o disparo remove instantaneamente $w(p, tr_i)$ fichas de cada um dos lugares de entrada e coloca fichas nos lugares de saída (TSAI et al., 1995).

A *Time Petri Net* (TPN) é uma extensão da PN proposta por P. M. Merlin e D. J. Farber em (FARBER; MERLIN, 1976) com o objetivo de representar características temporais em uma PN, e é um dos modelos mais utilizados para a modelagem e verificação de sistemas de tempo real (BERTHOMIEU; DIAZ, 1991). Uma TPN é definida como uma tupla $R =< Pl, Tr, B, F, Mo, SIM >$ na qual $M_o = Pl \rightarrow \mathbb{N}$ é a função de marcação inicial e $SIM = Tr \rightarrow Q^* \times (Q^* \cup \infty)$ (Q^* é o conjunto de números racionais positivos) é um mapeamento chamado intervalo estático. Dois valores de tempo $a \leq b$ são associados a cada transição tr_i, sendo estes relativos ao momento em que a transição é habilitada. Assumindo-se que cada transição tr_i é habilitada no tempo absoluto τ_{abs} e assim mantida continuamente, então tr_i não irá disparar antes do instante de tempo $(\tau_{abs} + a)$, mas deve disparar antes ou no máximo no instante $(\tau_{abs} + b)$. Uma TPN com dois lugares e uma transição tr_2 com $a = 5$ e $b = 10$ é ilustrada na Figura 4.1.

Em uma TPN, a união de todos os valores de disparo que são possíveis a partir de uma determinada marcação é definido como uma

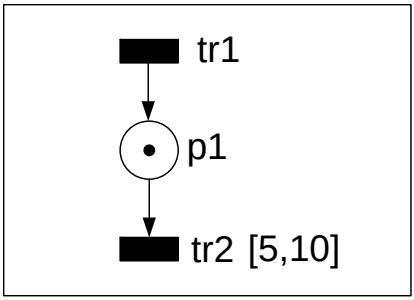

Figura 4.1: Exemplo de TPN

classe de estados (*state class*). A partir da classe de estados inicial pode ser construída uma árvore de classes que expressa o comportamento do sistema. Quando esta árvore tem um número limitado de nós distintos, um grafo finito pode ser associado à TPN. Este grafo é chamado de grafo de alcançabilidade (*reachability graph*) da TPN (BERTHOMIEU; DIAZ, 1991).

Mais informações sobre TPN podem ser encontradas, entre outros, em (BERTHOMIEU; DIAZ, 1991; NASPOLINI, 2006).

4.2.1.2 Linguagens de Especificação

De acordo com (NASPOLINI, 2006), na metodologia do *model checking* uma linguagem de especificação deve cumprir o requerimento de ser lógica ou temporal. Uma lógica temporal emprega proposições atômicas para realizar afirmações sobre os estados. Estas proposições são relações elementares as quais, em um dado estado, possuem um valor verdadeiro. As duas linguagens mais difundidas na literatura são a Lógica Temporal Linear (*Linear Temporal Logic*, LTL) e a *Computational Tree Logic* (CTL). Nesta proposta será empregada a CTL.

A lógica temporal CTL pode expressar convenientemente propriedades importantes do sistema em análise. Na CTL todas as fórmulas fazem referência a estados, e estas fórmulas especificam comportamentos do sistema a partir de um dado estado (EMERSON, 1991). Na sintaxe da linguagem, as proposições atômicas são fórmulas que podem ser

construídas combinando-se um quantificador de caminho *universal* (A) ou *existencial* (E). A fórmula quantificadora do caminho é verdadeira se e somente se a fórmula para o caminho é verdadeira para *todos* (*all*, A) ou *algum* (*some*, E) caminho completo que inicia no estado a partir da qual a fórmula está sendo avaliada.

As fórmulas de caminho são do tipo *globalmente* (*globally*, *G(p)*), *eventualmente* (*eventually F(p)*), *próximo estado* (*next-state X(p)*) ou *até que* (*until [p U q]*), sendo p e q fórmulas CTL. A fórmula *G(p)* é verdadeira se e somente se for verdadeira para *todos* os estados no caminho subsequente. A fórmula de caminho *F(p)* é verdadeira se e somente se for verdadeira em algum dos estados no caminho subsequente (ou seja, no caminho subsequente existe ao menos um estado no qual a fórmula é verdadeira). Um exemplo de fórmula CTL é $EF(p)$, significando "'Existe um caminho no qual p é verdadeiro"'.

4.2.1.3 *The Roméo Toolbox*

O Roméo Toolbox é um conjunto de softwares para a análise de TPNs, desenvolvido no *Real-Time Systems Team* no *L'Institut de Recherche en Communications et Cybernetique de Nantes* (IRCCyN).

Pode realizar análises em TPN e em algumas de suas extensões, sendo que, entre outras funcionalidades, pode computar o grafo de classes de estados de Redes de Petri, traduzir TPNs para autômatos temporizados, verificar propriedades de alcançabilidade durante a construção do grafo (*on-the-fly verification*) e simular graficamente uma TPN.

A verificação *on-the-fly* permite a checagem de propriedades sem a necessidade do grafo de alcançabilidade completo, o que pode ser interessante em casos nos quais a rede não é limitada, como modelos de tempo real onde *deadlines* podem ser perdidos (GARDEY et al., 2005).

Maiores informações sobre o Roméo podem ser encontradas em (Real-Time Systems Team at L'Institut de Recherche en Communications et Cybernetique de Nantes, 2010).

4.2.2 Método Proposto

A proposta deste trabalho é modelar o segmento dinâmico do FlexRay como uma TPN, sendo que cada mensagem dinâmica do sistema em análise é replicada a partir de um modelo padrão. No modelo completo é realizada uma verificação com o objetivo de checar a possibilidade de perdas de *deadline* por alguma das mensagens. O método proposto será detalhado no restante desta seção.

4.2.2.1 Modelo de Sistema

Este trabalho considera que a análise de escalonabilidade será realizada em sistemas em que o conjunto de mensagens dinâmicas e os tamanhos para $gdCycle$, ST_{bus} e DYN_{bus} são fornecidos. Apesar da especificação do FlexRay permitir a associação de um mesmo *slot* com mais de uma mensagem este trabalho considera que um *slot* estará associado com uma única mensagem, ou seja, cada mensagem possui um $FrameID$ único. Também se considera que a atribuição dos $FrameIDs$ das mensagens é fornecido.

As mensagens do segmento dinâmico devem ser periódicas, sem *offset* e com *deadline* menor ou igual ao período ($D \leq P$). O método proposto não aborda mensagens esporádicas devido ao indeterminismo associado às mesmas, pois este indeterminismo pode tornar a análise inviável devido ao tempo necessário para a construção do grafo de alcançabilidade da TPN.

O tempo de transmissão $C_m(MS)$ de uma mensagem m é obtido pela conversão do tamanho de m em bytes para inteiros de MS (Equação 4.1, sendo $BusSpeed$ a velocidade do canal de comunicação em bit/s). Considera-se que C_m contém os bytes referentes ao cabeçalho e trail para um quadro FlexRay.

O tamanho do segmento estático (em segundos) é convertido para inteiros de MS (Equação 4.2). O número $gNumberOfMinislots$ de MS que representa o tamanho do segmento dinâmico é dado pela Equação 4.3.

O período P_m é convertido para MS através da Equação 4.4. Por motivos que serão explicados na Seção 4.2.2.3 o *deadline* D_m é obtido através da Equação 4.5.

Os segmentos de controle NIT e *Symbol Window* não são explicitamente modelados porque ambos podem ser representados como uma mensagem estática ou como as mensagens de mais alta prioridade no segmento dinâmico, tendo período igual ao tamanho do ciclo FlexRay.

$$C_m(MS) = \left\lceil \frac{\text{tamanho de } m \text{ em bytes} \times 8}{BusSpeed \times gdMinislot} \right\rceil \tag{4.1}$$

$$ST_{bus}(MS) = \left\lceil \frac{ST_{bus}}{gdMinislot} \right\rceil \tag{4.2}$$

$$gNumberOfMinislots = \left\lceil \frac{DYN_{bus}}{gdMinislot} \right\rceil \tag{4.3}$$

$$P_m(MS) = \left\lceil \frac{P_m}{gdMinislot} \right\rceil \tag{4.4}$$

$$D_m(MS) = \begin{cases} \left\lceil \frac{D_m}{gdMinislot} \right\rceil & \text{se} \quad D_m < P_m \\ \left\lceil \frac{D_m}{gdMinislot} \right\rceil - 1 & \text{se} \quad D_m = P_m \end{cases} \tag{4.5}$$

4.2.2.2 Modelo Para um Ciclo FlexRay

A análise de mensagens estáticas não faz parte do escopo deste trabalho, por isso o segmento estático é representado no modelo proposto como sendo um único lugar ST que é o lugar de entrada de uma transição tr_{ST} cujo tempo de disparo é igual a $ST_{bus}(MS)$.

Uma correta análise do segmento dinâmico deve considerar não apenas as mensagens dinâmicas (cuja transmissão é arbitrada de acordo com o contador $vSlotCounter$) como também o número de MS disponíveis. No modelo proposto, a saída da transição tr_{ST} coloca no lugar $MSCounter$ (que representa o contador de MS) um número de fichas equivalente ao valor $gNumberOfMinislots$ do sistema. A saída de tr_{ST} também coloca uma ficha em um lugar que representa o primeiro *slot* do segmento dinâmico. O incremento de $vSlotCounter$ será expli-

cado na Seção 4.2.2.3. Se após o processamento de todos os $FrameID's$ o contador $MSCounter$ ainda possui fichas (ou seja, existem MS não utilizados no final do DYN) uma transição $tr_{removeDYN}$ remove estas fichas uma a uma. Após o final do segmento dinâmico, a transição tr_{DYN} coloca uma ficha no lugar ST, dando início a um novo ciclo FlexRay. Um exemplo do modelo (sem mensagens dinâmicas) é exibido na Figura 4.2(a).

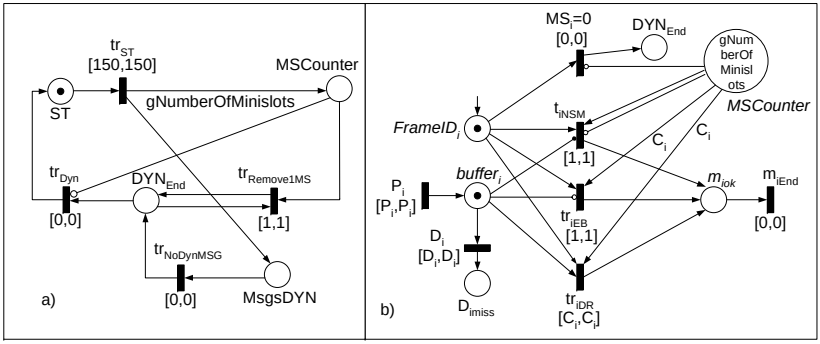

Figura 4.2: a) Exemplo de Modelo para o barramento. b) Modelo padrão para mensagens dinâmicas

4.2.2.3 Modelo para uma mensagem dinâmica

Um dos objetivos desta proposta é oferecer um método em que as características do segmento dinâmico sejam corretamente capturadas. Por este motivo, um modelo TPN que reproduz os possíveis comportamentos do segmento dinâmico foi elaborado. Como explicado no Capítulo 2, quando um nó é autorizado a transmitir uma mensagem no segmento dinâmico pode ocorrer uma das seguintes situações:

1. Existem dados para serem enviados e existem MS em número suficiente para a transmissão. Neste caso o *slot* utilizará o número necessário de MS para a transmissão dos dados.

2. Existem dados para serem enviados mas não existem MS disponíveis para a transmissão do quadro completo. Neste caso o *slot*

utiliza apenas um MS e a transmissão dos dados é postergada para o ciclo seguinte.

3. Não existem dados a serem enviados em um *slot*. Neste caso o *slot* utiliza apenas um MS.

4. O contador de MS fica vazio ($MSCounter = 0$) ou $vSlotCounter$ atinge $cSlotIDMax$. Neste caso o segmento dinâmico é finalizado e é iniciado um novo ciclo FlexRay.

O comportamento acima resulta em um modelo TPN para mensagens dinâmicas que é ilustrado na Figura 4.2(b). Este modelo será explicado a seguir.

Considere-se uma mensagem dinâmica m. Para esta mensagem, um lugar $buffer_i$ (sendo i o número do $FrameID$ associado a m) indica a existência de dados prontos para a transmissão. A transição P_i representa o período de m. A transição D_m representa o *deadline* de m, sendo disparada caso o *deadline* de m seja perdido, o que é indicado por uma ficha no lugar $D_{i_{miss}}$.

Um lugar $FrameID_i$ representa o número do *slot* atualmente em $vSlotCounter$. Este lugar recebe uma ficha da transição t_{ST} (caso em que é o primeiro $FrameID$ do sistema) ou da transição de saída $m_{i_{end}}$ de uma mensagem de prioridade mais alta. Quando uma ficha é colocada no lugar $FrameID_i$ pode ocorrer uma das seguintes situações (que estão associadas com os possíveis comportamentos descritos anteriormente:

1. Não existem MS disponíveis no segmento (ou seja, $MSCounter = 0$) (situação 4 descrita anteriormente). Neste caso, a transição $MS_i = 0$ é disparada, colocando o token diretamente no lugar que indica o final do segmento dinâmico. A transição $MS_i = 0$ é instantânea e não influencia no tempo de resposta do sistema.

2. Existem dados para serem enviados mas não MS em número suficiente para acomodar o quadro completo (situação 2 descrita anteriormente). Neste caso uma ficha é mantida no lugar $buffer_i$

e apenas um MS é retirado de $MScounter$. A transição $tr_{i_{NSM}}$ associada com esta ação tem um tempo de disparo igual a uma unidade de tempo e coloca uma ficha no lugar $m_{i_{ok}}$ indicando que a mensagem foi processada.

3. Não existem dados para envio no *slot* (situação 3). Como no caso anterior, apenas um MS é utilizado, sendo que a transição $tr_{i_{EB}}$ retira uma ficha de $MSCounter$ e coloca outra no lugar $m_{i_{ok}}$, mantendo a ficha em $buffer_i$.

4. Existem dados para transmissão e suficientes MS em $MSCounter$. A transição $tr_{i_{DR}}$ retira o número necessário de fichas do lugar $MSCounter$, a ficha de $buffer_i$ e a ficha de $FrameID_i$, colocando uma ficha no lugar $m_{i_{ok}}$ indicando assim que a mensagem foi processada. A transição $tr_{i_{DR}}$ tem um intervalo de disparo igual a $C_m(MS)$.

Por fim, a transição $m_{i_{end}}$ retira a ficha do lugar $m_{i_{ok}}$ e a coloca no lugar que indica o próximo $FrameID$ do segmento dinâmico (ou no lugar DYN_{End} se estiver associada com a mensagem de mais baixa prioridade) indicando assim o incremento de $vSlotCounter$. A transição $m_{i_{end}}$ é instantânea e não influencia no tempo de resposta de m.

O modelo TPN para uma mensagem dinâmica é ilustrado na Figura 4.2(b).

4.2.2.4 Modelo para uma rede completa

A TPN na qual a análise é realizada é composta por uma instância do modelo para o ciclo FlexRay apresentado na Seção 4.2.2.2 e uma instância do modelo de mensagem dinâmica (apresentado na Seção 4.2.2.3) para cada mensagem do sistema. Um arco conecta a transição tr_{ST} ao lugar $FrameID_1$ (uma mensagem com $FrameID = 1$ é a que possui a prioridade mais alta no sistema e está associada ao primeiro *slot* do segmento dinâmico). Outro arco conecta a transição $m_{1_{end}}$ ao lugar $FrameID_2$ da mensagem com a segunda prioridade mais alta e

assim sucessivamente, até que todas as instâncias de mensagens estejam interligadas. A transição $m_{i_{end}}$ da mensagem de mais baixa prioridade é conectada ao lugar DYN_{end}. O modelo TPN completo para um pequeno sistema FlexRay com três mensagens dinâmicas é ilustrado na Figura 4.3.

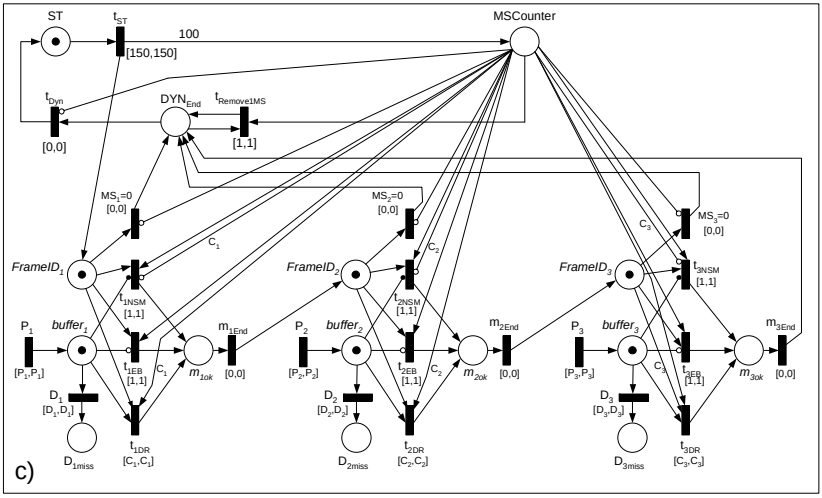

Figura 4.3: Exemplo de modelo TPN para um sistema FlexRay

Para garantir que uma TPN construída com o método proposto captura as características do segmento dinâmico do FlexRay foram feitas simulações com diferentes conjuntos de mensagens. O comportamento observado na simulação da TPN foi comparado com aquele esperado de um barramento FlexRay, atestando-se desta forma a validade do modelo.

4.2.3 Análise de Escalonabilidade

A análise de escalonabilidade de um sistema com o método aqui proposto é realizada verificando-se uma propriedade específica: a ausência de perdas de *deadline* por uma mensagem.

Como no modelo proposto a perda de um *deadline* é representada por uma ficha em um lugar $D_{i_{miss}}$ (sendo i o número do $FrameID$

associado com a mensagem m), a perda do *deadline* de m pode ser expressa pela fórmula CTL $EF(D_{i_{miss}} > 0)$. Esta fórmula pode ser interpretada como "eventualmente no futuro, o lugar $D_{i_{miss}}$ terá uma ou mais fichas", significando que, se em um ou mais dos possíveis caminhos do grafo de alcançabilidade da TPN ocorrer a perda de *deadline*, o lugar $D_{i_{miss}}$ terá mais que zero fichas. Obviamente esta propriedade não pode ser verdadeira em um sistema escalonável e, portanto, a resposta para a verificação deve ser "falso".

A verificação das propriedades em uma TPN é realizada sobre seu grafo de alcançabilidade. Entretanto, em sistemas grandes o esforço requerido para a construção deste grafo pode requerer um longo (e potencialmente ilimitado) tempo de computação. Além disso, um sistema representado com o modelo apresentado neste documento e que seja não escalonável pode resultar em um grafo de alcançabilidade infinito devido ao ilimitado número de fichas que podem ser colocadas no lugar $D_{i_{miss}}$, tornando a análise virtualmente impossível.

Neste trabalho, os problemas relacionados à construção do grafo de alcançabilidade são minimizados através do uso de algumas técnicas. A primeira destas técnicas é verificar se existe a perda do *deadline* de uma mensagem durante a construção do grafo, parando o processo se uma determinada propriedade é satisfeita (por exemplo, na perda do *deadline* por uma mensagem).

A segunda técnica é limitar o tamanho do grafo de alcançabilidade pela limitação do intervalo de verificação. Entretanto, um problema é definir este intervalo. Considerando que todas as mensagens neste trabalho são periódicas, se for assumido que a primeira chegada de todas as mensagens é no instante $t = 0$, então todas as mensagens chegarão juntas novamente em um instante equivalente ao valor do mínimo múltiplo comum (MMC) entre os períodos das mensagens. Como para todas as mensagens o *deadline* é menor ou igual ao período ($D \leq P$), no instante equivalente a este valor todos os *deadlines* já deverão ter sido cumpridos. Assim sendo, o valor τ do MMC entre períodos das mensagens pode ser utilizado como o instante final da

verificação, resultando na fórmula CTL $EF[0,\tau](D_{i_{miss}} > 0)$.

No modelo proposto, dependendo do número de mensagens dinâmicas no instante $t = \tau$, pode haver uma explosão combinatória no grafo de alcançabilidade, tornando a análise impossível. Este problema pode ser evitado pela adição de um pequeno pessimismo no sistema: para cada mensagem, o *deadline* é modificado para que se torne menor que o período ($D < P$). Esta modificação é feita pela subtração de 1MS do valor nominal de D_m caso $D_m = P_m$, o que resulta na Equação 4.5 apresentada anteriormente. Como o período permanece o mesmo, a modificação implica em que todos os *deadlines* devem ser cumpridos antes de τ. Isto significa que a verificação da TPN pode ser parada imediatamente antes do instante de tempo dado por τ, pois todos os *deadlines* já deverão ter sido cumpridos. No modelo proposto todos os valores de tempo são inteiros de MS (Seção 4.2.2.1)e por isso a proposta é parar a verificação 1MS antes de τ, o que leva a fórmula CTL abaixo:

$$EF[0,\tau-1](D_{i_{miss}} > 0) \quad \text{(I)}$$

Em um sistema com um número n de mensagens dinâmicas, a verificação pode ser realizada simultaneamente para todas as mensagens, resultando na fórmula CTL

$$EF[0,\tau-1]((D_{1_{miss}} > 0) \vee ... \vee (D_{n_{miss}} > 0)) \quad \text{(II)}$$

Um resultado "verdadeiro" (ou "true") para as Fórmulas CTL I e II significa que uma das mensagens perdeu seu *deadline*. Por outro lado, um resultado "falso" (ou "false") indica que o sistema é escalonável, sendo este o resultado esperado para a verificação. No método aqui proposto, as Fórmulas CTL I e II são utilizadas para a verificação das TPNs construídas de acordo com o método explicado na Seção 4.2.2.

4.2.4 Resultados Experimentais

Com o objetivo de demonstrar a viabilidade do método proposto, o tempo de computação requerido para a análise de escalonabilidade de sistemas com conjuntos de 10, 50, 70, 100, 150 e 220 mensagens foi medido. Todos os sistemas utilizados para medição possuem os mesmos valores para $gdCycle$, ST_{bus}, DYN_{bus} e $gdMinislot$, sendo estes apresentados na Tabela 4.2.

Parâmetro	Valor
$gdCycle$	5 ms
ST_{bus}	3 ms
DYN_{bus}	2 ms
$gdMinislot$	6.875μs
$gNumberOfMinislots$	290 MS

Tabela 4.2: Parâmetros utilizados para medições de tempo

Apesar de ser possível projetar uma TPN diretamente com o editor gráfico do Roméo Toolbox, devido ao grande número de lugares, transições e arcos necessários para representar um sistema construído com o método proposto na Seção 4.2.2 existe o risco potencial de ocorrências de erros durante a elaboração. Por isso, um *software* escrito em C++ foi desenvolvido para automatizar o processo. Este *software* tem como entrada um arquivo no qual são descritas as características da rede e das mensagens, e fornece como saída um arquivo no formato utilizado pelo Roméo Toolbox.

A Tabela 4.3 apresenta os tempos necessários para a análise de sistemas com 10, 50, 70, 100, 150 e 220 mensagens.

Os testes foram realizados em um computador com processador Intel(R) Xeon(R) 2.00GHz, 3.2GB de memória RAM e Linux Debian 4.3.2 com *kernel* versão 2.6.29, e com o Roméo Toolbox versão 2.9.0.

4.2.5 Considerações

O objetivo desta proposta é apresentar uma alternativa viável para a análise de escalonabilidade de mensagens no segmento dinâ-

Número de Mensagens	Tempo
10	<1s
50	6s
70	13s
100	33s
150	3min
220	8min

Tabela 4.3: Tempos necessários para a análise de diversos conjuntos de mensagens

mico do FlexRay. Entretanto, devido ao modelo de mensagens adotado (mensagens periódicas sem *jitter*), o método proposto tem utilidade limitada na prática, especialmente se for considerado o fato de que o segmento dinâmico do FlexRay é projetado para uso com mensagens esporádicas (SCHMIDT; SCHMIDT, 2008b). Por isso, serão apresentados a seguir três métodos heurísticos para obtenção de limites superiores para o tempo de resposta de mensagens no segmento dinâmico do FlexRay.

4.3 Heurística 1

A primeira heurística proposta neste trabalho é uma modificação daquela intitulada "Heuristic Solution for $BusCycles_m$" apresentada em (POP et al., 2008) [3].

A proposta da heurística original é reduzir o problema da definição de quantos ciclos de comunicação podem ser completamente ocupados por mensagens e *minislots* não utilizados com prioridade mais alta que uma mensagem m sob análise a um problema de empacotamento, tornando assim a análise mais simples, embora pessimista. Mas como discutido na Seção 3.2.2, dependendo do número e tamanho das mensagens do sistema, o pessimismo introduzido pode ser alto. Naquela proposta considera-se que $lf(m)$ é o conjunto das mensagens com prioridade maior que m e $lf(m,t)$ é o conjunto de todas as mensagens com

[3] A solução proposta em (POP et al., 2008) é abordada na Seção 3.2.2 deste documento.

prioridade mais alta que m com possível ocorrência dentro de um intervalo de tempo t. O tempo de comunicação das mensagens em $lf(m,t)$ é modificado de acordo com a Equação 3.17 gerando um novo conjunto $lf'(m,t)$ que é posteriormente submetido às equações apresentadas na Seção 3.2.2.3, resultando em um número $BusCycles(lf'(m,t))$ de ciclos cheios. O valor de $BusCycles(lf'(m,t))$ é utilizado na Equação 3.7 de onde, por fim, se obtém o tempo de resposta de m.

Nesta seção é apresentado um método onde se propõe utilizar o mesmo conjunto de equações apresentadas em (POP et al., 2008) e que resultam no valor de $BusCycles(lf'(m,t))$ modificando-se no entanto o modo de se definir o conjunto $lf'(m,t)$ e o tamanho mínimo a ser preenchido em cada ciclo. O objetivo destas modificações é reduzir o pessimismo introduzido no número de $BusCycles(lf'(m,t))$. O método proposto é apresentado a seguir.

O método considera que a análise de escalonabilidade será realizada em sistemas nos quais mensagens esporádicas são alocadas no segmento dinâmico de um barramento FlexRay. As mensagens possuem *deadline* menor ou igual ao período ($D \leq P$), sendo o tempo de transmissão C, o período P e o *deadline* D de cada mensagem dados em inteiros de MS. Também se considera que os tamanhos para $gdCycle$, ST_{bus} e $gNumberOfMinislots$ são fornecidos.

Assim como no trabalho original, considere-se m uma mensagem dinâmica sob análise, sendo $lf(m)$ o conjunto das mensagens com prioridades mais altas que m, n o número de mensagens em $lf(m)$ e $lf(m,t)$ um conjunto contendo todas as possíveis ocorrências dos elementos de $lf(m)$ no intervalo t. O número de ocorrências de cada mensagem $l \in lp(m)$ é dado por $\lceil (J_l + t)/T_l \rceil$ (T_l é o período da mensagem l e J_l seu *jitter*).

O primeiro passo do método é reservar em cada ciclo *minislots* equivalentes ao número de mensagens em $lf(m)$. Esta reserva é feita para garantir a propriedade de que no segmento dinâmico do FlexRay, se não houverem dados a serem transmitidos em um *slot*, este tem o tamanho de um único MS. Considera-se DYN'_{bus} o novo tamanho para

o segmento dinâmico, dado pela Equação 4.6:

$$DYN'_{bus} = gNumberOfMinislots - n. \tag{4.6}$$

O segundo passo é gerar um novo conjunto de mensagens $lf'(m,t)$ onde no tempo de transmissão C'_i de cada mensagem $i \in lf(m,t)$ é descontado o valor de 1 MS (Equação 4.7). Este ajuste é realizado para compensar o MS que foi reservado para cada mensagem no passo anterior (se não há a transmissão de dados, o *slot* tem o tamanho de um único MS. No entanto, se os dados são transmitidos, este MS é efetivamente utilizado para a transmissão).

$$C_i^{'} = C_i - 1, \forall i \in lf(m,t) \tag{4.7}$$

O terceiro passo é definir o novo valor para a capacidade mínima requerida para preencher um segmento dinâmico. Conforme descrito na Seção 3.2.2.2 a capacidade mínima para preencher um ciclo é dada por $pLatestTx_m \times gdMinislot$. Nesta proposta, a capacidade mínima passa a ser relativa ao valor de DYN'_{bus}, de forma que:

$$pLatestTx'_m = DYN'_{bus} - C'_m \tag{4.8}$$

sendo C'_m o tempo de transmissão da mensagem m sob análise do qual foi descontado 1MS.

Por fim, $lf'(m,t)$ e $pLatestTx'_m$ são utilizados em conjunto com a heurística para "*bin packing*" apresentada na Seção 3.2.2.2, de onde se obtém o valor de $BusCycles(lf'(m,t))$ que por sua vez é aplicado na Equação 3.7 obtendo-se assim um limite superior para o *wcrt* de uma mensagem m.

O método proposto tem complexidade pseudo-polinomial, e a qualidade da solução proposta será discutida no Capítulo 5 onde são feitas comparações entre os diversos métodos de análise presentes neste documento.

4.4 Heurística 2

A segunda solução heurística proposta neste documento tem como base a equação para análise de sistemas FPS apresentada em (AUDSLEY et al., 1993). Segundo aquele trabalho o tempo de resposta R_i de uma tarefa i que possui tempo de transmissão C_i e período T_i é dado por:

$$R_i = C_i + I_i \tag{4.9}$$

O valor de I_i representa a máxima interferência causada por mensagens com prioridade maior que i em qualquer intervalo de tempo $[t, t + R_i)$ e é dado por:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_i}{T_j} \right\rceil C_j \tag{4.10}$$

sendo J_j o *release jitter* e T_j o período de uma tarefa $i \in hp(i)$. Tarefas esporádicas podem ser incorporados ao modelo assumindo-se que o período T é igual ao seu intervalo mínimo entre chegadas (*Minimum Interarrival Time*, MIT).

A proposta apresentada nesta seção parte da premissa de que é possível estender a Equação 4.10 para que a mesma possa ser empregada na análise de mensagens dinâmicas do FlexRay, a exemplo do que é feito para o CAN em (DAVIS et al., 2007).

O método aqui proposto considera que a análise de escalonabilidade será realizada em sistemas nos quais mensagens esporádicas são alocadas no segmento dinâmico de um barramento FlexRay. As mensagens possuem *deadline* menor ou igual ao período ($D \leq P$), sendo o tempo de transmissão C, o período P e o *deadline* D de cada mensagem dados em inteiros de MS. Também se considera que os tamanhos para $gdCycle$, ST_{bus} e $gNumberOfMinislots$ são fornecidos.

Para facilitar o entendimento da proposta, considere-se como exemplo o sistema apresentado na Tabela 4.4. Neste sistema, todas as

mensagens são esporádicas, os valores de tempo são dados em MS e se deseja obter o tempo de resposta da mensagem m_3. O pior cenário possível para a transmissão de m_3 é ilustrado na Figura 4.4, onde pode-se observar que a transmissão de m_3 é atrasada por vários fatores: instâncias do segmento estático que ocorrem no intervalo da análise, mensagens com prioridade mais alta que m_3 e MS não utilizados. Pode-se observar também que m_3 não é transmitida no Ciclo 2 por não existir um mínimo de $pLatestTx_3$ MS disponíveis.

Parâmetro	**Valor em MS**
FC	15
ST_{bus}	4
$gNumberOfMinislots$	11

Mensagem	***C***	***MIT***	***D***
m_1	6	100	100
m_2	5	100	100
m_3	5	100	100

Tabela 4.4: Exemplo de Sistema FlexRay

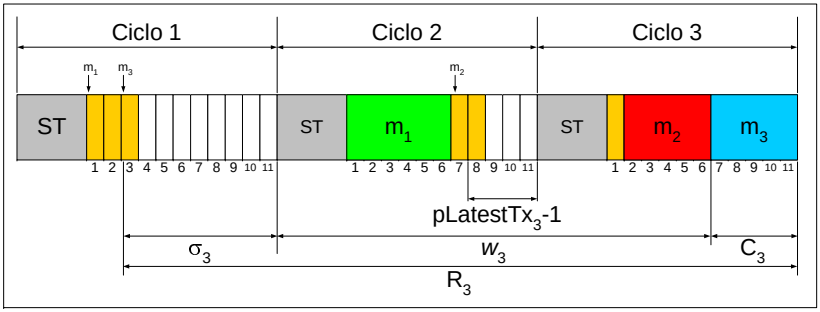

Figura 4.4: Exemplo de sistema para Heurística 2

No método que será apresentado nesta seção propõe-se que os fatores que podem atrasar a transmissão de uma mensagem do segmento dinâmico sejam incorporados à Equação 4.10, de forma a se obter um limite superior para o tempo de resposta R_i de uma mensagem i. Estes fatores serão descritos abaixo, sendo apresentadas também as equações

relacionadas aos mesmos. Por fim, será apresentada a equação completa para a heurística proposta.

Como discutido em outros capítulos deste documento, o pior caso para uma mensagem no FlexRay é ser gerada pela tarefa emissora imediatamente após o início do *slot* associado ao seu *FrameID*, pois desta forma a mensagem precisará esperar até o próximo ciclo para realmente concorrer pelo barramento. Este comportamento torna possível representar R_i como sendo

$$R_i = \sigma_i + w_i + C_i \tag{4.11}$$

sendo σ_i o tempo decorrido entre a chegada da mensagem no controlador até o início do próximo ciclo, w_i o atraso causado por outros fatores e C_i o tempo de transmissão da mensagem. Se considerado que por "início do próximo ciclo" se entende "início do próximo ciclo FlexRay", então σ_i é dado por:

$$\sigma_i = gdCycle - (ST_{bus} + (FrameID_i - 1) \times gdMinislot). \tag{4.12}$$

O atraso w_i é a soma dos demais fatores que interferem na transmissão de uma mensagem dinâmica i, que são:

Interferência causada pelo Segmento Estático: O FlexRay é um protocolo com arbitragem baseada em ciclos compostos por segmentos de tamanho fixo que se repetem a intervalos definidos. Se for considerado que o segmento estático é sempre o primeiro segmento de um ciclo FlexRay, e que o valor de σ_i apresentado anteriormente é o intervalo entre a chegada de i e o início do próximo ciclo FlexRay, então qualquer intervalo $\Delta - \sigma_i > 0$ contém ao menos uma fração de segmento estático que interfere na transmissão de mensagens dinâmicas, como pode ser observado na Figura 4.4. Nesta proposta, a interferência causada por instâncias de segmento estático é nomeada w_{ST} e seu valor

dado por:

$$w_{ST} = \left\lfloor \frac{R_i}{gdCycle} \right\rfloor \times ST_{bus} + \min\left(ST_{bus}, R_i - \left\lfloor \frac{R_i}{gdCycles} \right\rfloor \times gdCycle\right). \quad (4.13)$$

Como pode ser deduzido da equação, cada ciclo FlexRay inteiro no intervalo de análise resulta em um valor correspondente a um segmento estático inteiro, sendo que na fração do intervalo de análise correspondente ao último ciclo caso seja necessário é atribuído um valor equivalente a uma fração do ST.

Interferência causada por MS não utilizados com prioridade maior que *i*: No segmento dinâmico do FlexRay, um *slot* associado a um $FrameID$ tem tamanho de 1 MS caso não seja utilizado para a transmissão de dados. É fácil observar que em qualquer ciclo FlexRay uma mensagem i pode ser atrasada por no máximo $FrameID_i - 1$ MS devido a *slots* de prioridade mais alta nos quais não foram transmitidos dados, mas o valor exato de MS em cada ciclo é um problema combinatório cuja solução varia de sistema para sistema, sendo que nem mesmo existem garantias de que um *slot* no qual não foram transmitidos dados tenha o tamanho de 1MS (por exemplo, se no sistema da Figura 4.4 existisse uma mensagem $m4$ com $C_4 = 5$, no terceiro ciclo não haveria espaço para a transmissão de 1MS pelo *slot* associado ao seu $FrameID$).

Neste trabalho, o conjunto de MS inutilizados devido a não transmissão de mensagens por *slots* com prioridade mais alta que i é denominado $ms(i)$, e uma simplificação é feita em relação ao mesmo: ao invés de se tentar obter um valor exato, no início de cada segmento dinâmico se reserva 1 MS por mensagem do sistema, compensando esta reserva no tempo de transmissão de cada mensagem. Esta simplificação reduz a complexidade do problema, mas introduz um pessimismo no resultado do tempo de resposta de i. O impacto deste pessimismo será discutido mais adiante.

Como os MS reservados no início de cada segmento dinâmico no intervalo de análise atrasam a transmissão de i, esta reserva passa a ser considerada como a interferência w_{MS} cujo valor é dado por:

$$w_{MS} = \left\lfloor \frac{R_i}{gdCycle} \right\rfloor \times nm + \min\left\{ nm, \max\left(0, R_i - \left\lfloor \frac{R_i}{gdCycle} \right\rfloor \times gdCycle - ST_{bus}\right)\right\} \tag{4.14}$$

sendo nm o número de mensagens no sistema com prioridade igual ou superior a i. Como pode ser deduzido da Equação 4.14 cada ciclo FlexRay inteiro no intervalo de análise resulta em uma reserva de MS equivalente ao número de mensagens e, na fração correspondente ao último ciclo é reservado o total de MS apenas se houver espaço, caso contrário a reserva tem o tamanho disponível.

Interferência causada pela utilização mínima no DYN: Em um segmento dinâmico do FlexRay a utilização mínima que é necessária para evitar a transmissão de uma mensagem i equivale a $gNumberOfMinislots - (pLatestTx_i - 1)$ MS. Isto pode ser observado na Figura 4.4 em que, no Ciclo 2 a transmissão de m_3 é impedida pela falta de apenas 1MS.

A heurística aqui proposta assume que em cada fração do intervalo de análise equivalente a um segmento dinâmico será gasto apenas o mínimo necessário para garantir que i seja postergada para o ciclo seguinte. Com o objetivo de garantir esta premissa, $pLatestTx_i - 1$MS são considerados uma interferência que ocorre a cada intervalo correspondente a um ciclo FlexRay, de forma que a interferência causada pela utilização mínima do segmento dinâmico é dada por:

$$w_{UM} = \left\lceil \frac{(R_i + C_i)}{gdCycle} \right\rceil \times (C_i - 1). \tag{4.15}$$

Nesta equação o valor de C_i no segundo termo sofre a correção de 1MS devido àquele já reservado pela Equação 4.14.

Interferência por mensagens de prioridade mais alta que *i*: Outro fator de interferência são as mensagens com prioridade mais alta que i. Esta interferência já compõe a equação original utilizada para sistemas FPS (Equação 4.10), por isso a mesma é mantida nesta proposta, sendo apenas definido o valor para o *release jitter*. Seu valor é dado por:

$$w_{hpi} = \sum_{j \in lf(i)} \left\lceil \frac{R_i + (gNumberofMinislots - j)}{T_j} \right\rceil \times (C_j - 1) \quad (4.16)$$

em que $(gNumberofMinislots - j)$ representa o *release jitter* J_j de cada mensagem com prioridade mais alta que i. No valor do tempo de transmissão de cada mensagem em $lf(i)$ é compensado o MS reservado anteriormente.

A equação completa utilizada para calcular a interferência sofrida por uma mensagem i é composta pela soma de todos os fatores que podem interferir na transmissão de i sendo portanto a interferência w_i dada por:

$$w_i = w_{ST} + w_{MS} + w_{UM} + w_{hpi}. \quad (4.17)$$

4.4.1 Exemplo Numérico

Como exemplo numérico para o método proposto nesta seção, considere-se um sistema FlexRay com os parâmetros apresentados na Tabela 4.5, onde C é o tempo de transmissão das mensagens e MIT é o intervalo de tempo mínimo entre as chegadas das mensagens.

Os tempos de transmissão para cada mensagem calculados por este método bem como o tempo de computação necessário para o cálculo são apresentados na Tabela 4.6. A heurística foi implementada em C++ sendo executada em um computador com processador Intel(R) Xeon(R) 2.00GHz, 3.2GB de memória RAM e Linux Debian 4.3.2 com *kernel* versão 2.6.29.

Parâmetros	
ST_{bus}	10
gNumberOfMinislots	10

Mensagem	C	MIT
m_1	3	1000
m_2	3	38
m_3	3	55
m_4	4	1000

Tabela 4.5: Parâmetros para exemplo com a Heurística 2

Mensagem	Tempo de Resposta
m_1	22
m_2	24
m_3	26
m_4	65
Tempo de computação:	0,06s

Tabela 4.6: Tempos de resposta calculados com a Heurística 2

4.4.2 Considerações

A solução proposta nesta seção considera a suposição de que, em um dado sistema FlexRay em que mensagens esporádicas são alocadas no segmento dinâmico o pior caso para uma mensagem dinâmica m sob análise é ser postergada devido a falta de apenas 1MS. Como já demonstrado na Seção 4.1 esta premissa nem sempre é válida se utilizada se utilizada de forma isolada, mas devido à existência de outras fontes de pessimismo o risco do método aqui proposto apresentar resultados otimistas para o valor do tempo de resposta no pior caso de uma mensagem sob análise é descartado.

O método proposto tem complexidade pseudo-polinomial, e a qualidade da solução proposta será discutida no Capítulo 5 onde são feitas comparações entre os diversos métodos de análise presentes neste documento.

4.5 Heurística 3

A terceira heurística proposta neste trabalho é uma variação daquela apresentada na Seção 4.4 e utiliza um algoritmo para a obtenção de limites superiores para o tempo de resposta de mensagens esporádicas alocadas no segmento dinâmico do FlexRay.

O algoritmo calcula de forma iterativa o número de ciclos FlexRay completos que são necessários para acomodar o atraso sofrido por uma mensagem dinâmica m sob análise, sendo que cada iteração corresponde a um ciclo FlexRay. No método proposto, em cada iteração é analisado se existe entre o serviço oferecido e a demanda do segmento dinâmico uma folga que permita o envio de uma mensagem m sob análise. Caso exista esta folga, o *wcrt* de m é calculado. Caso contrário, é descontado da demanda total um valor mínimo e a demanda remanescente é novamente analisada na iteração seguinte, onde são atualizados os valores para a demanda e serviço oferecido. Este procedimento é realizado até que m possa ser transmitida ou até que o valor calculado indique a perda do *deadline* de m.

A seguir são dadas as definições para os valores do serviço oferecido e valor mínimo descontado da demanda sendo logo após apresentado e explicado o algoritmo proposto.

O serviço oferecido $sOferecido$ é o total de MS oferecidos a cada ciclo, do qual são descontados o tamanho do segmento estático ST_{bus} e uma reserva rMS equivalente a 1MS por mensagem dinâmica no sistema. Assim como em outras propostas neste trabalho, a reserva rMS é devido a uma simplificação adotada para evitar o problema combinatório causado por *minislots* associados a *slots* dinâmicos não utilizados para transmissão de dados em um dado ciclo, e depende da prioridade da mensagem m sob análise. O valor do serviço oferecido a cada ciclo é dado por

$$sOferecido = gdCycle - (ST_{bus} + rMS). \tag{4.18}$$

O valor minimo que em cada iteração é descontado da demanda

total é outra simplificação adotada nesta proposta. Considera-se que, se em uma iteração a demanda de MS no segmento dinâmico for maior que o serviço oferecido, da demanda total é descontado apenas a mínima demanda em MS que seria necessária para evitar a transmissão de m naquele ciclo. Como basta a falta de 1MS no segmento dinâmico para que m não possa ser transmitida, o valor da demanda mínima é dado por:

$$dMinima = gNumberOfMinislots - (C_m - 1 - 1) \tag{4.19}$$

sendo que nesta equação também é compensado em C_m o MS reservado por rMS.

Um atraso considerado no algoritmo é aquele que uma mensagem pode sofrer caso perca seu *slot* de transmissão. Sendo que o controlador de comunicação de um nó FlexRay decide se uma mensagem será enviada para o barramento no início do seu *slot*, o pior caso para a mensagem é chegar no controlador imediatamente após o início do *slot*, o que a obrigará a esperar até o início do próximo ciclo para realmente concorrer pelo barramento. Para uma mensagem m com $FrameID_m$ este atraso σ_m (em MS) é dado por:

$$\sigma_m = gdCycle - (ST_{bus} + FrameID_m - 1). \tag{4.20}$$

Para a explicação do algoritmo proposto (exibido em Algoritmo 1), considere-se um sistema FlexRay onde S é o conjunto de mensagens dinâmicas, n é o número de mensagens em S, m é a mensagem cujo tempo de resposta no pior caso R_m se deseja obter, $FrameID_m$ é o identificador de m e $hp(m)$ é o conjunto de mensagens com prioridade mais alta que m. Os valores S, n, m, $sOferecido$, $dMinima$, σ_m e os tamanhos para o ciclo FlexRay, segmento estático e segmento dinâmico são passados como entrada deste algoritmo

O algoritmo utiliza duas variáveis auxiliares atualizadas a cada iteração, sendo uma para armazenar o número de ciclos completos (CComp) e outra para armazenar o valor da demanda de transmis-

Algoritmo 1: Algoritmo para Heurística 3

Entrada: S, n, m, $sOferecido$, $dMinima$, σ_m, $gdCycle$, $gNumberOfMinislots$, ST_{bus}
Saída: R_m

```
1  início
2      CComp ← 0;
3      dTxAtendida ← 0;
4      enquanto Verdadeiro faça
5          intervalo ← CComp ×gdCycle + ST_bus;
6          dTxTotal ← Σ_{j∈hp(m)} ⌈((intervalo+j)+Jitter_j)/T_j⌉ × (C_j − 1);
7          dTxAtual ← dTxTotal − dTxAtendida;
8          folgaNoCiclo ← sOferecido − dTxAtual;
9          se folgaNoCiclo ≤ C_m − 1 − 1 então
10             imprimir "Não existe folga suficiente no ciclo";
11             CComp ← CComp + 1;
12             dTxAtendida ← CComp × dMinima;
13             se (σ_m + CComp ×gdCycle + C_m) > D_m então
14                 imprimir "m perdeu seu Deadline";
15                 retorna erro;
16             fim se
17         fim se
18         senão
19             imprimir "Existe folga suficiente no ciclo";
20             ocupadoUC ← gdCycle - (folgaNoCiclo - (C_m − 1));
21             R_m ← σ_m + CComp ×gdCycle + ocupUC;
22             se R_m > D_m então
23                 imprime "m perdeu seu Deadline";
24                 retorna erro;
25             fim se
26             senão
27                 retorna R_m;
28             fim se
29         fim se
30     fim enquanto
31 fim
```

são já atendida (dTxAtendida). No início do algoritmo, estas variáveis tem seu valor definido como zero (linhas 2 e 3 em Algoritmo 1).

O método considera que a análise de escalonabilidade será realizada em sistemas nos quais mensagens esporádicas são alocadas no segmento dinâmico de um barramento FlexRay. As mensagens possuem *deadline* menor ou igual ao período ($D \leq P$), sendo o tempo de transmissão C, o período P e o *deadline* D de cada mensagem dados em inteiros de MS. Também se considera que os tamanhos para $gdCycle$, ST_{bus} e $gNumberOfMinislots$ são fornecidos.

Para a demanda dinâmica total existente em um intervalo de tempo, o algoritmo considera um limite superior dado pela soma dos tempos de transmissão de todas as instâncias de mensagens dinâmicas em $hp(m)$ que podem ser geradas naquele intervalo. Este limite superior para a demanda é dado pela Equação 4.21 (note que no tempo de transmissão de cada mensagem é compensado o MS reservado em rMS). Os valores para $dTxTotal$ e para o intervalo de tempo são atualizados a cada iteração (linhas 5 e 6 em Algoritmo 1).

$$dTxTotal = \sum_{j \in hp(m)} \left\lceil \frac{(intervalo + j) + Jitter_j}{T_j} \right\rceil \times (C_j - 1) \quad (4.21)$$

A cada iteração é recalculada a demanda de transmissão atual $dTxAtual$. Este valor considera a demanda total no intervalo de tempo decorrido e a demanda já atendida em ciclos anteriores, de forma que seu valor é dado por $dTxAtual = dTxTotal - dTxAtendida$ (linha 7).

O valor de $dTxAtual$ é utilizado para calcular a folga existente no ciclo atual ($folgaNoCiclo$). Esta folga é obtida descontando-se do serviço oferecido a demanda atual (linha 8).

Por fim, é feita uma comparação para verificar se no ciclo atual existe espaço suficiente para a transmissão de m. Se não existir (linha 9) o número de ciclos ocupados é incrementado (linha 11), o valor da demanda atendida é atualizado (considerando-se o valor de $dMinima$ apresentado na Equação 4.19, linha 12), é verificado se existe a possibi-

lidade de m já haver perdido seu *deadline* (evitando-se assim a execução de iterações desnecessárias, linha 13) e finalmente iniciada uma nova iteração. Caso exista folga no ciclo (linha 18) é calculado o tempo de resposta R_m (linha 21) e, caso R_m seja menor ou igual a D_m, retornado este valor como resposta.

O método proposto tem complexidade pseudo-polinomial, e a qualidade da solução proposta será discutida no Capítulo 5 onde são feitas comparações entre os diversos métodos de análise presentes neste documento.

4.5.1 Exemplo Numérico

Como exemplo numérico para o método proposto nesta seção, considere-se um sistema FlexRay com os parâmetros apresentados na Tabela 4.7.

Parâmetros	
ST_{bus}	10
gNumberOfMinislots	10

Mensagem	**C**	**MIT**
m_1	3	1000
m_2	3	38
m_3	3	55
m_4	4	1000

Tabela 4.7: Parâmetros para exemplo com a Heurística 3

Os tempos de transmissão para cada mensagem calculados por este método bem como o tempo de computação necessário para o cálculo são apresentados na Tabela 4.8. O algoritmo foi implementado em C++, sendo executado em um computador com processador Intel(R) Xeon(R) 2.00GHz, 3.2GB de memória RAM e Linux Debian 4.3.2 com *kernel* versão 2.6.29.

Mensagem	Tempo de Resposta
m_1	22
m_2	24
m_3	26
m_4	65
Tempo de computação:	0,06s

Tabela 4.8: Tempos de resposta calculados com a Heurística 3

4.6 Conclusões

Neste capítulo foram apresentadas quatro propostas de métodos para a análise de escalonabilidade no segmento dinâmico do FlexRay, sendo que três delas calculam o tempo de resposta para mensagens esporádicas alocadas neste segmento.

As propostas apresentadas são resumidas na Tabela 4.9 onde é descrita a abordagem utilizada e os tipos de mensagens suportados por cada proposta (se periódicas, esporádicas ou ambas).

Como pode ser observado na tabela, todos os métodos propostos para a análise de sistemas com mensagens esporádicas alocadas no segmento dinâmico são heurísticas que fornecem um limite superior para o tempo de resposta das mensagens. O fato das propostas apresentadas serem apenas métodos heurísticos é um reflexo da dificuldade de propor um método que encontre o pior cenário para a transmissão de mensagens esporádicas alocadas no segmento dinâmico. Como discutido na Seção 4.2.2 obter o valor exato para o pior cenário de transmissão é um problema combinatório classificado como NP-completo pela teoria computacional, requerendo um elevado, se não inviável, custo de computação.

No próximo capítulo serão apresentadas avaliações experimentais da qualidade de cada solução proposta.

Proposta	Abordagem	Tipos de mensagem	Comentários
Análise do DYN com uso de TPN	Análise com modelo em TPN	Periódicas	O modelo de mensagens é restritivo
Heurística 1	Método heurístico	Periódicas e esporádicas	É baseado na Heurística proposta em (Pop et al., 2008)
Heurística 2	Método heurístico	Periódicas e esporádicas	É baseado na equação para FPS proposta em (Burns and Wellings, 2001)
Heurística 3	Método heurístico	Periódicas e esporádicas	É baseado no número de ciclos completos necessários para a transmissão da mensagem sob análise

Tabela 4.9: Resumo sobre os artigos relacionados à análise do tempo de resposta no DYN

Capítulo 5

Avaliação Experimental

No capítulo anterior foram apresentadas quatro propostas de métodos para a análise do tempo de resposta de mensagens alocadas no segmento dinâmico do FlexRay.

Com o objetivo de avaliar a qualidade destas propostas este capítulo apresenta experimentos onde as mesmas são comparadas com os métodos da literatura descritos no capítulo 3. São avaliadas a eficácia e a eficiência de cada proposta.

O método da proposta "Análise do DYN Com Redes de Petri" (seção 4.2) não será incluído nos experimentos por suportar apenas mensagens periódicas. Avaliações deste método são apresentadas na própria descrição da proposta (seção 4.2.4).

5.1 Descrição dos Experimentos

O objetivo dos experimentos é comparar as técnicas propostas no capítulo 4 com os métodos existentes na literatura. Como discutido no capítulo 3, o único método da literatura para o qual não foi possível encontrar um cenário que leve a resultados otimistas para o tempo de resposta das mensagens foi o método heurístico proposto em (POP et al., 2008), sendo por isso este método utilizado como *benchmark* nas

avaliações.

Os métodos propostos foram avaliados em relação a quatro cenários compostos por sistemas que representam diferentes combinações para os tamanhos das mensagens. Os parâmetros dos cenários foram definidos a partir do que é usual em sistemas automotivos atuais.

A seguir serão apresentadas as definições utilizadas para os sistemas e para os cenários.

5.1.1 Definições para os sistemas

Os experimentos consideram sistemas que utilizam um barramento FlexRay com velocidade de 10Mb/s e um canal de comunicação. A este barramento estão ligadas ECUs que geram mensagens esporádicas que estão alocadas no segmento dinâmico.

A discussão e avaliação de métodos para definição dos parâmetros de um sistema que utilize o protocolo FlexRay não é parte do escopo deste trabalho, por isso os testes realizados utilizam valores arbitrários para o período das mensagens, tamanho $gdMinislot$ dos *minislots* dinâmicos, tamanho $gdCycle$ do ciclo FlexRay, tamanho dos segmentos estático e dinâmico, tempo de transmissão e número de mensagens dinâmicas. Apesar de arbitrários, estes parâmetros foram definidos de forma a representar valores existentes em sistemas reais, pois não existe sentido em analisar sistemas irreais. A definição para cada parâmetro bem como uma explicação sobre a mesma será dada a seguir.

Período das mensagens: O período escolhido para as mensagens dinâmicas teve como base um veículo FIAT Stilo. O FIAT Stilo utiliza redes CAN onde as mensagens do domínio do *chassis* tem períodos entre 10 e 100 ms (FIATFORUM, 2010). Com o objetivo de diversificar os testes realizados, foram escolhidos períodos de 10, 20, 50, 100 e 200 ms. No entanto, em alguns cenários não é possível obter a utilização do barramento desejada apenas com estes valores, de forma que quando necessário foram utilizados também períodos de 5ms e 500ms.

$gdMinislot$: Para o tamanho de cada *minislot* dinâmico foi utilizado o valor de 6,875 μs. Em um barramento de 10Mb/s, este valor

equivale a 8 *bytes* de dados. O valor foi escolhido principalmente por ser aquele utilizado pela BMW (SCHEDL, 2007).

gdCycle: O tamanho de um ciclo FlexRay foi definido como 5ms. Este tamanho equivale à metade do menor período utilizado para as mensagens (10ms). Além disso, é o valor utilizado pela BMW em sistemas com FlexRay (SCHEDL, 2007).

Tamanho do segmento estático: O segmento estático utilizado nos testes foi definido como tendo 1ms. Um intervalo de 1ms em um barramento com velocidade de 10Mb/s possibilitaria a transmissão de 156 mensagens de 8 bytes, o dobro do número de mensagens CAN presentes no FIAT Stilo (FIATFORUM, 2010).

Tamanho do segmento dinâmico: O valor $gNumberOfMinislots$ do segmento dinâmico é a diferença entre o tamanho do ciclo FlexRay e o tamanho do segmento estático, tendo portanto 4ms de duração. Considerando-se o tamanho definido para cada *minislot*, o segmento dinâmico é composto por 582MS.

Tempos de transmissão das mensagens dinâmicas: Considerar nos experimentos um tempo de transmissão igual a 1MS equivaleria a tornar o problema uma simples análise TDMA. Por isso, foram definidos três valores para os tempos de transmissão: 2MS, 4MS e 30MS. Um pacote FlexRay com tamanho de 2MS em um barramento com velocidade de 10Mb/s é suficiente para transmitir o *payload* de um pacote CAN no formato *standard* (incluindo o cabeçalho e a cauda do pacote FlexRay). O valor de 4MS equivale ao dobro do primeiro valor, e 30MS foi um valor escolhido para representar mensagens grandes no sistema, sendo que este valor pouco menor que um pacote FlexRay com o *payload* máximo utilizado.

Número de mensagens dinâmicas: O número de mensagens dinâmicas em cada sistema foi definido como 400 mensagens.

Todas as mensagens possuem *deadline* igual ao mínimo intervalo entre chegadas ($D = MIT$). Considera-se que cada mensagem de um sistema está associada a um $FrameID$ único, sendo os $FrameIDs$ atribuídos de acordo com o *deadline* de cada mensagem (*deadline mono-*

tonic: quanto menor do *deadline*, maior a prioridade e portanto menor o $FrameID$).

Como estão sendo avaliados métodos relacionados ao segmento dinâmico, mensagens alocadas no segmento estático não são consideradas, sendo utilizado apenas o parâmetro do tamanho do segmento estático.

5.1.2 Definições para os cenários

Os métodos propostos foram avaliados em relação a quatro cenários que representam diferentes combinações para os tamanhos das mensagens.

Em cada cenário foram feitas análises para utilizações do segmento dinâmico que variam de 30 a 80%, com tolerância de 1 p.p. Para cada faixa de utilização foram analisados 50 sistemas com 400 mensagens. Os cenários utilizados são descritos a seguir.

Cenário 1: Composto por sistemas homogêneos em que todas as mensagens possuem tempo de transmissão $C = 2MS$. Este cenário representa sistemas compostos apenas por mensagens pequenas, como por exemplo sistemas em que o FlexRay é utilizado como *backbone* entre *gateways* CAN.

Cenário 2: Composto por sistemas homogêneos em que todas as mensagens possuem tempo de transmissão $C = 30MS$. Este cenário representa sistemas compostos apenas por mensagens grandes, como por exemplo sistemas em que o FlexRay é utilizado como *backbone* entre *gateways* CAN, mas sendo as mensagens agrupadas antes das transmissões entre os *gateways*.

Cenário 3: Composto por sistemas heterogêneos onde 50% das mensagens possuem tempo de computação $C = 2MS$ e 50% possuem $C = 30MS$. Representa uma situação hipotética onde o sistema é composto igualmente por mensagens grandes e pequenas.

Cenário 4: Composto por sistemas heterogêneos onde 45% das mensagens possuem tempo de computação $C = 2MS$, 45% possuem $C = 4MS$ e 10% possuem $C = 30MS$. Representa sistemas mistos

compostos por muitas mensagens pequenas e poucas mensagens grandes.

Com o objetivo de garantir uma variação nos experimentos, no caso dos sistemas heterogêneos os tempos de computação foram distribuídos de forma aleatória entre as mensagens (respeitando os limites do número de mensagens associado a cada tempo de computação).

De forma similar à distribuição dos tempos de computação nos sistemas heterogêneos, os possíveis períodos para mensagens (descritos na seção 5.1.1) foram associados de forma aleatória às mensagens de cada sistema. Sobre o conjunto destas mensagens foram realizados ajustes adequando a utilização para os limites fixados para cada avaliação, sendo posteriormente o conjunto de mensagens organizado de acordo com o *deadline* para a atribuição dos $FrameIDs$.

5.1.3 Critérios de avaliação

Para cada cenário de análise foram avaliadas a eficácia e a eficiência dos métodos propostos.

A avaliação da eficácia mede a relação entre os resultados obtidos e os objetivos pretendidos. No caso dos experimentos apresentados neste documento, a eficácia é a relação entre o número de sistemas indicados como escalonáveis por cada método proposto e a quantidade de sistemas analisados. Entretanto, como não existe na literatura um método exato para a análise de escalonabilidade de sistemas com mensagens esporádicas alocadas no segmento dinâmico do FlexRay que possa ser utilizado como *benchmark*, não é garantido que os sistemas utilizados nas análises realmente sejam escalonáveis, o que pode influenciar nos resultados obtidos.

A avaliação da eficiência refere-se ao tempo de computação necessário para as análises nos diversos cenários. Neste caso é apresentado o tempo médio de execução observado durante a análise de 50 sistemas.

Os experimentos foram executados em um computador DELL Precision 690, com dois processadores Intel(R) Xeon(R) 2.00GHz *dual core*, 3.2GB de memória RAM e Linux Debian 4.3.2 com *kernel* versão

2.6.29.

5.2 Resultados dos Experimentos

Nesta seção são apresentados e discutidos os resultados obtidos nos experimentos realizados para cada cenário.

5.2.1 Resultados do Cenário 1

A Figura 5.1 apresenta a eficácia da heurística utilizada como *benchmark* e dos métodos propostos para o Cenário 1. Pode-se observar nesta figura que para utilizações entre 20 e 50% do segmento dinâmico as três heurísticas propostas neste trabalho indicam que 100% dos sistemas são escalonáveis. Para a utilização de 60% apenas a Heurística 2 indicou sistemas escalonáveis, conseguindo atingir um índice de 100%. Para utilizações acima de 60% nenhuma das heurísticas conseguiu indicar a existência de sistemas escalonáveis.

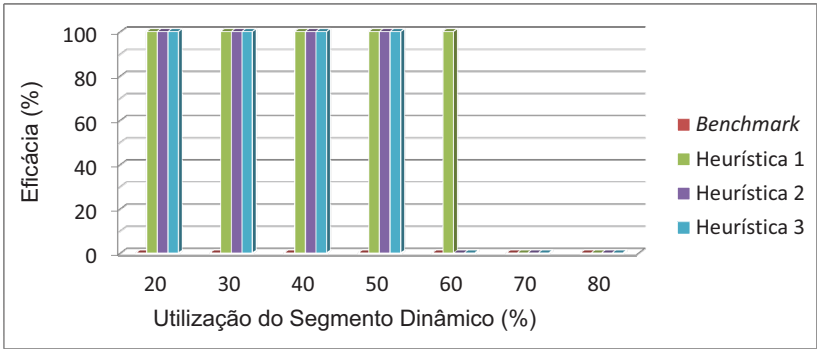

Figura 5.1: Eficácia para o Cenário 1

Um comentário importante sobre os resultados ilustrados na Figura 5.1 é que a heurística utilizada como *benchmark* não conseguiu indicar a existência de sistemas escalonáveis para nenhuma das situações. Desta forma, a heurística apresentada em (POP et al., 2008) mostrou-se inútil.

Outro ponto que chama a atenção é a brusca variação na eficácia das propostas para as utilizações de 50, 60 e 70%. Como pode ser observado na figura, a eficácia das Heurísticas 2 e 3 cai de 100% a 0% entre a utilização de 50 e 60%, o mesmo ocorrendo para a Heurística 1 com as utilizações de 60 e 70%.

A Figura 5.2 apresenta a eficiência das heurísticas para o Cenário 1. Pode-se observar que o tempo de computação utilizado pelo *benchmark* é consideravelmente mais alto que o utilizado pelas heurísticas propostas, mesmo não indicando a existência de sistemas escalonáveis no cenário.

Para as utilizações entre 20 e 50% do segmento dinâmico, as heurísticas 2 e 3 requerem um tempo de computação muito mais baixo que aquele requerido pela Heurística 1, obtendo no entando os mesmos resultados.

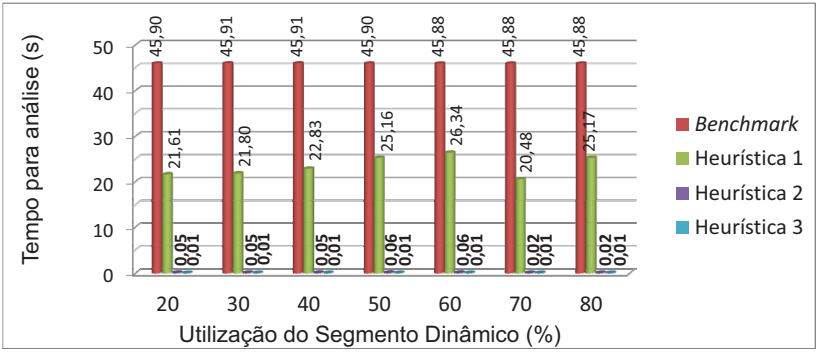

Figura 5.2: Eficiência para o Cenário 1

Na Tabela 5.1 são apresentados os valores mínimos e máximos para o tempo de computação requerido por cada heurística. Pode-se verificar que a variação no tempo de computação é pequena.

Na Tabela 5.2 é apresentado o desvio padrão para o tempo de computação medido em cada faixa de utilização do Cenário 1.

		Heurística			
		Benchmark	**1**	**2**	**3**
Utilização 20%	**Min**	45,47s	21,24s	0,04s	0,01s
	Max	46,14s	22,37s	0,06s	0,01s
Utilização 30%	**Min**	45,43s	21,36s	0,04s	0,01s
	Max	46,13s	22,12s	0,06s	0,01s
Utilização 40%	**Min**	46,19s	23,25s	0,04s	0,01s
	Max	45,69s	22,28s	0,06s	0,01s
Utilização 50%	**Min**	45,42s	24,64s	0,05s	0,01s
	Max	47,01s	26,22s	0,06s	0,01s
Utilização 60%	**Min**	45,64s	25,86s	0,06s	0,01s
	Max	46,10s	27,15s	0,07s	0,01s
Utilização 70%	**Min**	45,50s	20,01s	0,02s	0,01s
	Max	46,08s	21,46s	0,03s	0,01s
Utilização 80%	**Min**	45,47s	24,62s	0,02s	0,01s
	Max	46,13s	25,88s	0,02s	0,01s

Tabela 5.1: Tempos de computação mínimo e máximo para as heurísticas no Cenário 1

	Heurística			
	Benchmark	**1**	**2**	**3**
Utilização 20%	0,10s	0,22s	0,005s	0,001s
Utilização 30%	0,12s	0,20s	0,005s	0,001s
Utilização 40%	0,09s	0,24s	0,003s	0,001s
Utilização 50%	0,20s	0,27s	0,001s	0,001s
Utilização 60%	0,09s	0,25s	0,002s	0,001s
Utilização 70%	0,11s	0,24s	0,003s	0,001s
Utilização 80%	0,12s	0,29s	0,001s	0,001s

Tabela 5.2: Desvio padrão para os tempos de computação no Cenário 1

5.2.2 Resultados do Cenário 2

A Figura 5.3 apresenta a eficácia das avaliações realizadas no Cenário 2. Para este cenário, as heurísticas propostas indicaram que 100% dos sistemas são escalonáveis para a utilização de 20%, mas apenas a Heurística 1 conseguiu indicar existência de sistemas escalonáveis para a utilização de 30% do segmento dinâmico. A heurística apresentada em (POP et al., 2008) não conseguiu indicar a presença de sistemas escalonáveis.

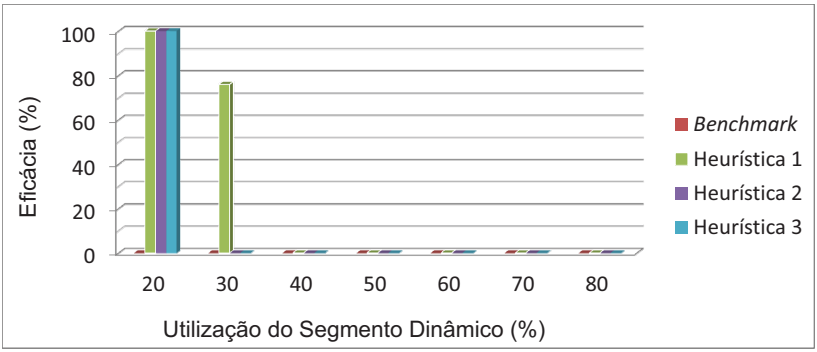

Figura 5.3: Eficácia para o Cenário 2

A eficiência dos métodos apresentados para o Cenário 2 é apresentada na Figura 5.4. Como pode ser observado, o tempo requerido pela Heurística 1 é bastante superior ao tempo requerido pelas Heurísticas 2 e 3, mas ainda assim é menor que o requerido pela heurística apresentada em (POP et al., 2008).

Na Tabela 5.3 são apresentados os valores mínimos e máximos para o tempo de computação requerido por cada heurística.

Na Tabela 5.4 é apresentado o desvio padrão para o tempo de computação medido em cada faixa de utilização do Cenário 2.

5.2.3 Resultados do Cenário 3

A eficácia das heurísticas propostas neste trabalho para o Cenário 3 é apresentada na Figura 5.5. Para este cenário, os três méto-

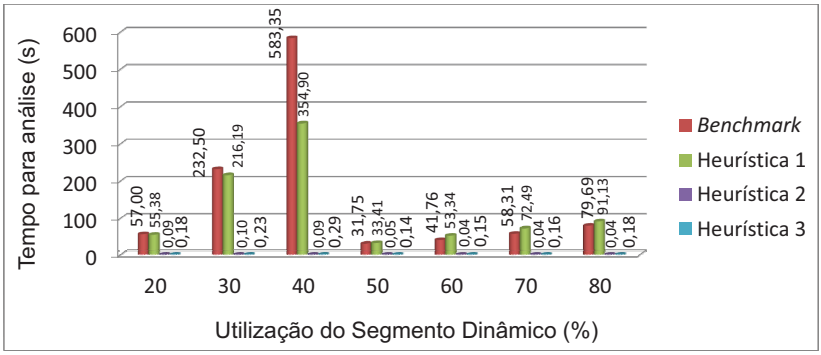

Figura 5.4: Eficiência para o Cenário 2

		Heurística			
		Benchmark	**1**	**2**	**3**
Utilização 20%	**Min**	52,10s	48,62s	0,09s	0,17s
	Max	63,30s	64,06s	0,11s	0,18s
Utilização 30%	**Min**	192,84s	177,79s	0,09s	0,22s
	Max	258,63s	234,54s	0,10s	0,28s
Utilização 40%	**Min**	506,49s	321,30s	0,08s	0,28s
	Max	631,47s	382,08s	0,10s	0,30s
Utilização 50%	**Min**	31,35s	33,13s	0,04s	0,14s
	Max	32,01s	34,13s	0,05s	0,15s
Utilização 60%	**Min**	40,12s	51,92s	0,04s	0,15s
	Max	43,09s	54,96s	0,05s	0,16s
Utilização 70%	**Min**	55,27s	67,50s	0,03s	0,16s
	Max	60,30s	74,48s	0,04s	0,17s
Utilização 80%	**Min**	76,55s	85,82s	0,03s	0,16s
	Max	82,38s	96,99s	0,04s	0,27s

Tabela 5.3: Tempos de computação mínimo e máximo para as heurísticas no Cenário 2

	Heurística			
	Benchmark	**1**	**2**	**3**
Utilização 20%	4,66s	6,69s	0,005s	0,001s
Utilização 30%	7,28s	5,45s	0,005s	0,010s
Utilização 40%	9,95s	9,44s	0,006s	0,007s
Utilização 50%	0,10s	0,21s	0,004s	0,002s
Utilização 60%	0,76s	0,67s	0,003s	0,005s
Utilização 70%	1,29s	1,49s	0,001s	0,004s
Utilização 80%	1,61s	3,42s	0,005s	0,005s

Tabela 5.4: Desvio padrão para os tempos de computação no Cenário 2

dos propostos conseguiram indicar a existência de sistemas escalonáveis para a utilização de 20 e 30% do segmento dinâmico do FlexRay, mas nenhum dos métodos avaliados consegue indicar a existência de sistemas escalonáveis para utilizações maiores. De forma similar ao ocorrido nos dois cenários anteriores, a heurística proposta em (POP et al., 2008) não conseguiu indicar a existência de sistemas escalonáveis para nenhuma das utilizações, sendo novamente considerada inútil para a análise neste cenário.

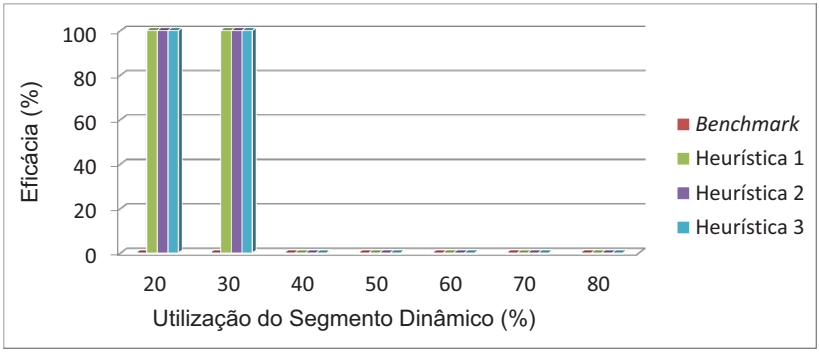

Figura 5.5: Eficácia para o Cenário 3

A eficiência dos métodos apresentados no Cenário 3 é apresentada na Figura 5.6. No gráfico, nota-se que para este cenário a Heurística 1 requer um tempo de computação muito mais elevado que o

requerido pelas demais heurísticas, ficando próximo dos 20 minutos para algumas utilizações.

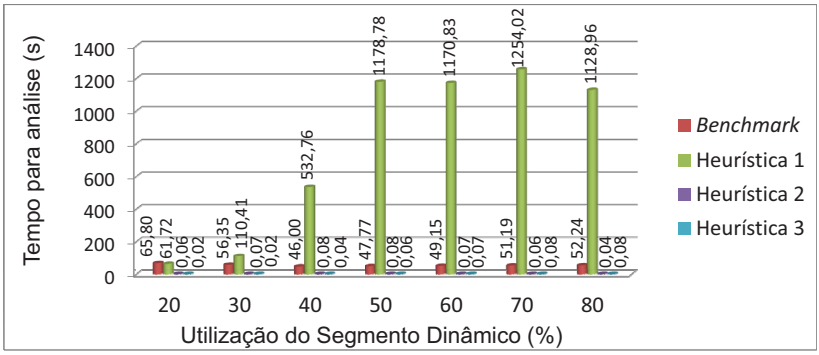

Figura 5.6: Eficiência para o Cenário 3

Na Tabela 5.3 são apresentados os tempos mínimos e máximos requeridos para as análises nas faixas de utilização definidas para este cenário. Pode-se observar que existe uma grande variação entre o maior e o menor tempo utilizado pela Heurística 1 na análise do cenário, variação que não é tão grande nas demais heurísticas.

Na Tabela 5.6 é apresentado o desvio padrão para o tempo de computação medido em cada faixa de utilização do Cenário 3.

5.2.4 Resultados do Cenário 4

A eficácia das heurísticas para o Cenário 4 é apresentada na Figura 5.7. Entretanto, para este cenário nenhum dos métodos avaliados conseguiu indicar a presença de sistemas escalonáveis para a faixa de utilizações originalmente definidas para os experimentos. Por este motivo, também foram avaliados para este cenário sistemas com uma utilização de 20%, sendo que para esta utilização as Heurísticas 1, 2 e 3 mostram que todos os sistemas são escalonáveis.

A heurística proposta em (POP et al., 2008) não indicou a presença de sistemas escalonáveis nem mesmo para a utilização de 20%, sendo inútil também para a análise deste cenário.

		Heurística			
		Benchmark	**1**	**2**	**3**
Utilização 20%	**Min**	60,45s	53,75s	0,05s	0,02s
	Max	71,77s	68,42s	0,08s	0,03s
Utilização 30%	**Min**	43,49s	92,05s	0,06s	0,02s
	Max	86,68s	131,76s	0,10s	0,03s
Utilização 40%	**Min**	43,06s	347,24s	0,07s	0,03s
	Max	47,86s	813,87s	0,10s	0,04s
Utilização 50%	**Min**	45,72s	837,36s	0,08s	0,06s
	Max	50,33s	1580,95s	0,10s	0,07s
Utilização 60%	**Min**	47,21s	895,65s	0,06s	0,06s
	Max	51,75s	1443,16s	0,08s	0,08s
Utilização 70%	**Min**	48,97s	1032,44s	0,05s	0,07s
	Max	54,30s	1574,25s	0,06s	0,08s
Utilização 80%	**Min**	49,15s	772,69s	0,03s	0,07s
	Max	55,01s	1447,84s	0,04s	0,08s

Tabela 5.5: Tempos de computação mínimo e máximo para as heurística no Cenário 3

	Heurística			
	Benchmark	**1**	**2**	**3**
Utilização 20%	2,86s	3,29s	0,005s	0,003s
Utilização 30%	3,22s	7,18s	0,007s	0,004s
Utilização 40%	0,82s	53,17s	0,006s	0,004s
Utilização 50%	0,91s	62,92s	0,006s	0,002s
Utilização 60%	1,09s	55,27s	0,005s	0,006s
Utilização 70%	1,32s	49,83s	0,005s	0,005s
Utilização 80%	1,36s	56,80s	0,001s	0,005s

Tabela 5.6: Desvio padrão para os tempos de computação no Cenário 3

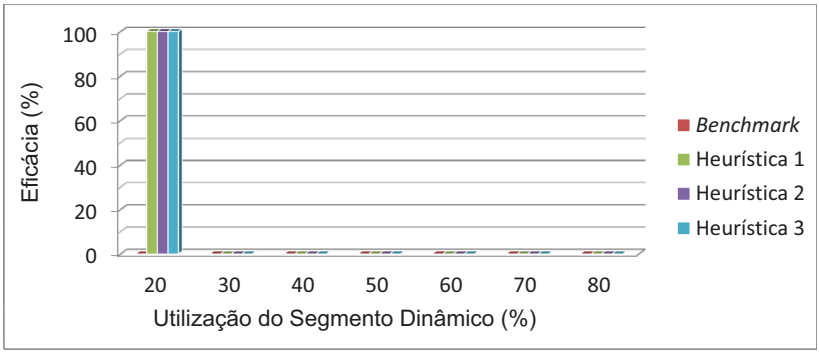

Figura 5.7: Eficácia para o Cenário 4

A eficiência dos métodos para Cenário 4 é apresentada na Figura 5.8. No gráfico, nota-se que o tempo de computação necessário para a heurística proposta em (POP et al., 2008) e para Heurística 1 é bastante alto para algumas situações. Já os tempos para as heurísticas 2 e 3 continuam baixos, a exemplo do ocorrido nos demais cenários.

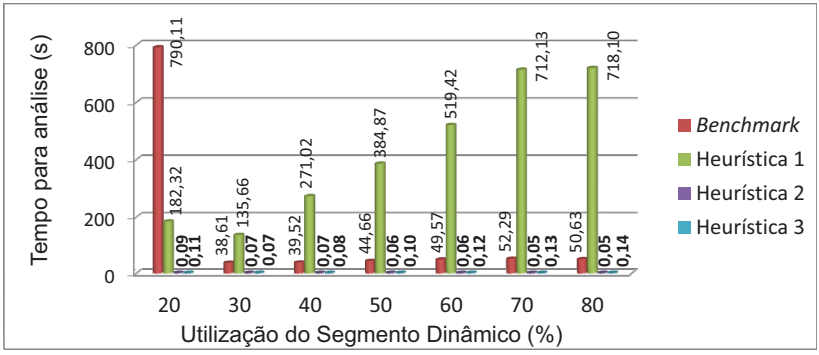

Figura 5.8: Eficiência para o Cenário 4

Os tempos de computação mínimo e máximo para cada heurística utilizada neste cenário são apresentados na Tabela 5.7. Assim como para o cenário anterior, pode-se observar a existência de uma grande variação entre o maior e o menor tempo utilizado pela heurística proposta em (POP et al., 2008) e pela Heurística 1, enquanto os tempos

necessários para as Heurísticas 2 e 3 mantém a diferença pequena.

		Heurística			
		Benchmark	**1**	**2**	**3**
Utilização 20%	**Min**	242,33s	112,51s	0,08s	0,1s
	Max	2565,99s	366,91s	0,10s	0,11s
Utilização 30%	**Min**	34,76s	116,68s	0,06s	0,06s
	Max	52,62s	220,04s	0,08s	0,07s
Utilização 40%	**Min**	35,94s	192,88s	0,06s	0,08s
	Max	48,04s	433,00s	0,08s	0,09s
Utilização 50%	**Min**	39,10s	243,21s	0,06s	0,10s
	Max	61,81s	714,79s	0,07s	0,11s
Utilização 60%	**Min**	43,57s	381,11	0,05s	0,11s
	Max	60,37s	796,58s	0,07s	0,12s
Utilização 70%	**Min**	46,33s	453,64s	0,05s	0,13s
	Max	57,46s	951,57s	0,06s	0,16s
Utilização 80%	**Min**	44,42s	455,26s	0,04s	0,14s
	Max	56,38s	1060,99s	0,06s	0,15s

Tabela 5.7: Tempos de computação mínimo e máximo para as heurística no Cenário 4

Na Tabela 5.8 é apresentado o desvio padrão para o tempo de computação medido em cada faixa de utilização do Cenário 4.

	Heurística			
	Benchmark	**1**	**2**	**3**
Utilização 20%	6,44s	3,16s	0,008s	0,005s
Utilização 30%	0,98s	1,82s	0,006s	0,002s
Utilização 40%	1,06s	1,50s	0,006s	0,005s
Utilização 50%	1,91s	1,43s	0,004s	0,002s
Utilização 60%	1,88s	1,39s	0,005s	0,002s
Utilização 70%	1,14s	2,30s	0,004s	0,004s
Utilização 80%	0,90s	1,80s	0,004s	0,001s

Tabela 5.8: Desvio padrão para os tempos de computação no Cenário 4

5.3 Comparação entre Cenários

Com base nos resultados dos experimentos realizados para os Cenários 1 a 4, é apresentada na Tabela 5.9 uma comparação qualitativa entre as heurísticas avaliadas.

Heurística	Eficácia	Eficiência
Heurística (POP et al., 2008)	Pior	Lento
Heurística 1	Melhor	Lento
Heurística 2	Intermediária	Rápido
Heurística 3	Intermediária	Rápido

Tabela 5.9: Comparação qualitativa para as heurísticas avaliadas

Através da tabela, pode-se verificar que a Heurística 1 apesar de lenta teve a melhor eficácia para os cenários considerados.

Já as Heurísticas 2 e 3 tem a melhor eficiência, mas sua eficácia é menor que a apresentada pela Heurística 1. Nos testes realizados em cada cenário (apresentados na seção 5.2), o índice de eficiência para estas duas heurísticas foi bastante similar.

Por fim, a heurística proposta em (POP et al., 2008) (utilizada como *benchmark*) teve as piores eficácia e eficiência entre os métodos avaliados. Isso ocorre principalmente pelo fato de que os parâmetros utilizados nos cenários dos experimentos não são favoráveis a esta heurística, devido ao pessimismo introduzido pela mesma. Para que o *benchmark* pudesse encontrar sistemas escalonáveis, seriam necessárias alterações em alguns parâmetros, como por exemplo, *deadline* e intervalo mínimo entre chegadas. Entretanto, algumas das mudanças necessárias poderiam fazer com que os cenários analisados ficassem distantes de sistemas utilizados em aplicações reais.

Embora vários parâmetros de cada cenário sejam arbitrários, eles foram escolhidos com base em sistemas reais da indústria automotiva,

por isso sua relevância.

5.4 Conclusões

Neste capítulo foram apresentados os resultados de experimentos realizados com os métodos heurísticos propostos no capítulo 4 deste documento.

Estes experimentos tiveram como objetivo avaliar a eficácia e a eficiência das propostas na análise de sistemas FlexRay onde mensagens esporádicas são alocadas no segmento dinâmico do FlexRay. Para as avaliações foram utilizados quatro cenários diferentes, sendo que estes cenários foram elaborados de forma a representar possíveis sistemas reais para veículos que utilizem o protocolo FlexRay. Em cada cenário, os métodos foram avaliados em relação a utilizações de 30, 40, 50, 60, 70 e 80% do segmento dinâmico.

Visando comparar os métodos propostos neste trabalho com as técnicas existentes na literatura, o método heurístico proposto em (POP et al., 2008) foi adotado como *benchmark* para os experimentos. Esta heurística foi escolhida por ser o único método para o qual, como discutido no capítulo 3, não foi possível elaborar um cenário em que o tempo de resposta calculado para mensagens esporádicas alocadas no segmento dinâmico do FlexRay não fosse otimista. Entretanto, este método apresentou um péssimo desempenho nos cenários utilizados, um resultado que já era esperado dada as conclusões sobre o mesmo apresentadas na seção 3.2.2.2.

Nos resultados dos experimentos, pode-se observar que a Heurística 1 proposta neste trabalho tem uma eficácia maior que as Heurísticas 2 e 3, mas a sua eficiência é menor, sendo estas últimas muito mais rápidas nas análises dos sistemas utilizados nas avaliações. Por isso, a proposta final apresentada neste documento é que as três heurísticas apresentadas no capítulo 4 sejam unificadas em um único método de análise.

Neste método, um sistema para o qual se deseja verificar a escalo-

nabilidade é inicialmente analisado pelas Heurísticas 2 e 3 (que possuem alta eficiência mas eficácia moderada). Se o sistema for considerado escalonável, a análise está concluída. Se o sistema for considerado não escalonável, o mesmo é analisado novamente pela Heurística 1 que possui baixa eficiência mas alta eficácia.

Finalmente, é preciso ter cautela quanto aos resultados das experiências para as situações em que os métodos propostos não conseguiram indicar a existência de sistemas escalonáveis. Como não existe um método exato para a análise de mensagens esporádicas alocadas no segmento dinâmico do FlexRay, não é possível garantir que os sistemas utilizados nos experimentos realmente sejam escalonáveis. Nos experimentos as heurísticas podem estar indicando como não escalonável um sistema que realmente não é escalonável, ou seja, a análise não está sendo pessimista.

O objetivo dos experimentos é comparar as heurísticas entre si (eficácia relativa entre elas), e não fazer afirmações sobre eficácia absoluta.

Capítulo 6

Conclusão

O Sistema de Comunicação FlexRay é protocolo de dados desenvolvido por uma aliança de fabricantes que inclui BMW, Daimler-Chrysler e Bosh, e tem sido promovido pela literatura como sendo o futuro padrão de fato em aplicações automotivas que requerem desempenho e confiabilidade, como os futuros sistemas *X-by-Wire*. Através da combinação de métodos estáticos e dinâmicos para a transmissão de mensagens, o FlexRay fornece, entre outros aspectos, flexibilidade, largura de banda e determinismo.

Um importante problema relacionado com o FlexRay é a análise do tempo de resposta de mensagens alocadas no segmento dinâmico, especialmente se estas mensagens forem esporádicas. Devido às características do protocolo, determinar o pior cenário possível para a transmissão de mensagens esporádicas no segmento dinâmico é um problema NP-completo cuja solução exige um alto custo computacional, tornando seu uso na prática inviável para grandes sistemas. Apesar de existirem na literatura métodos para esta análise, estes sofrem de falhas que, ou podem levar a resultados otimistas para determinados cenários, ou acrescentam demasiado pessimismo no resultado para tempo de resposta.

O objetivo desta dissertação de mestrado é aumentar o arsenal

teórico a disposição dos engenheiros que projetam sistemas com barramentos FlexRay no que diz respeito ao segmento dinâmico. Neste sentido, a dissertação inclui no capítulo 3 uma revisão bibliográfica crítica dos métodos existentes, e nos capítulos 4 e 5 a proposta e avaliação de novos métodos de análise.

Neste trabalho foram propostos quatro métodos alternativos para a análise de escalonabilidade de mensagens alocadas no segmento dinâmico FlexRay, sendo que três destes métodos são heurísticas para encontrar um limite superior para o tempo de resposta de mensagens esporádicas alocadas neste segmento.

A eficácia e a eficiência dos métodos heurísticos propostos foram avaliadas através de experimentos utilizados quatro cenários diferentes, sendo que estes cenários foram elaborados de forma a representar possíveis sistemas reais para veículos que utilizem o protocolo FlexRay.

O primeiro cenário utilizado nas avaliações é composto apenas por mensagens dinâmicas com tamanho de 2MS, representando sistemas onde existem apenas mensagens pequenas. No segundo cenário todas as mensagens tem tamanho de 30MS representando sistemas compostos apenas por mensagens grandes. No Cenário 3, 45% das mensagens tem tamanho de 2MS, 45% tem tamanho de 4MS e 10% tamanho de 30MS, representando sistemas compostos por mensagens de tamanhos variados. Por fim, no Cenário 4 50% das mensagens tem tamanho de 2MS e 50% tem tamanho de 30MS, representando cenários compostos apenas por mensagens pequenas e grandes. Em cada cenário, os métodos foram avaliados em relação a utilizações de 30, 40, 50, 60, 70 e 80% do segmento dinâmico. Como a definição de parâmetros para sistemas que utilizem o protocolo FlexRay não faz parte do escopo deste trabalho, para os experimentos foram utilizados parâmetros arbitrários que representam valores próximos a sistemas reais existentes atualmente.

Visando comparar as propostas deste trabalho com as técnicas existentes na literatura, o método heurístico proposto em (POP et al., 2008) foi adotado como *benchmark* nos experimentos realizados. Esta

heurística foi escolhida por ser o único método para o qual, como discutido no capítulo 3, não foi possível elaborar um cenário em que o tempo de resposta calculado para mensagens esporádicas alocadas no segmento dinâmico do FlexRay não foi otimista.

Através das avaliações realizadas chegou-se a conclusão de que, devido ao desempenho das heurísticas propostas, as mesmas podem ser unificadas em um mesmo método de análise no qual um sistema para o qual se deseja verificar a escalonabilidade é inicialmente analisado pelas Heurísticas 2 e 3 (que possuem alta eficiência mas eficácia moderada). Se o sistema for considerado não escalonável, o mesmo é analisado novamente pela Heurística 1 que possui baixa eficiência mas alta eficácia.

Um problema encontrado durante a elaboração deste trabalho é que o fato de não existir na literatura um método exato de análise para mensagens esporádicas alocadas no segmento dinâmico do FlexRay dificulta não só a avaliação de novos métodos propostos, mas também o próprio projeto de sistemas que empreguem o protocolo.

Uma sugestão de trabalho futuro é o desenvolvimento de um método exato para a análise do segmento dinâmico do FlexRay. Outra proposta é a integração das heurísticas propostas neste trabalho em um método mais abrangente que possibilite a análise heurística de um sistema veicular com redes de comunicação híbridas. Uma terceira sugestão é um trabalho que aprofunde o estudo sobre o desempenho das heurísticas propostas para, se possível, introduzir melhorias nas mesmas. Por fim, uma quarta sugestão é aprofundar o estudo do *Real-Time Calculus* e seu emprego no contexto de barramentos FlexRay.

Apêndice A

Implementação do Método ILP Proposto em (Pop et al., 2008)

```
/* Numero de mensagens no set                *
 * De acordo com artigo POP:  considerando n *
 * elementos em lf(m,t), existem ao menos n  *
 * bus cycles que podem ser utilizados       */
param nroMsg, integer, > 0;

/* Numero de minislots em um ciclo DYN       */
param gNumberOfMinislots, integer, > 0;
param cMsgEmAnalise, integer, >0;
param pLatestTxm, integer > 0;

/* Conjuntos                                 */
set I := 1..nroMsg;  /* set Ciclos           */
set J := 1..nroMsg;  /* set FrameID          */
set K := 1..nroMsg;  /* set Messages lf(m,t) */

/* Conjunto de mensagens lf(m,t)             */
param setLF{k in K}, integer, > 0;

/* Atribuicao de Frame ID                    *
```

```
 * Frame[j,k] eh uma constante binaria com   *
 * valor 1 se a mensagem k tem FrameID j     */
param Frame{j in J, k in K}, binary;

/* Variavel yi=1..n                          *
 * De acordo com artigo POP: variavel criada *
 * para cada bus cycle, setada para 1 se o   *
 * ciclo for preenchido, 0 caso contrario    */
var y{i in I}, binary;

/* Variavel x[i,j,k]: variavel binaria com   *
 * valor 1 se a mensagem k, com FrameID j eh *
 * enviada no ciclo i                        */
var x{i in I, j in J, k in K}, binary;

/* Restricoes                                */

/* Restricao (10): uma mensagem k eh enviada *
 * apenas uma vez                            *
 * O somatorio da tabela x[i,j,k] deve ser   *
 * igual a 1, ou seja, a mensagem eh         *
 * transmitida apenas uma vez                */
s.t. mensagemApenasUmaVez{k in K}: sum{i in I, j in J}
             x[i,j,k] = 1;

/* Restricao (11): cada FrameID eh utilizado *
 * apenas uma vez em um ciclo                *
 * O somatorio de  x[i,j,k] para todo i,j    *
 * deve ser ≤1: no maximo uma vez, mas      *
 * pode nao haver espaco para transmitir     */
s.t. umIDporCiclo{i in I, j in J}: sum{k in K}
             x[i,j,k] ≤ 1;

/* Restricao (12): cada mensagem eh          *
 * transmitida com o FrameID associado       */
s.t. mensagemFID{i in I, j in J, k in K}: x[i,j,k]
     ≤ Frame[j,k];

/* Condicoes (8) e (9):   a variavel binaria *
 * y[i] = 1 a carga do bus no ciclo i eh     *
 * eh maior que pLatestTx da mensagem sob    *
 * analise                                   */
s.t. load{i in I}: sum{j in J, k in K}
     (x[i,j,k] * setLF[k])
     + sum{j in J} (1 - sum{k in K}
     x[i,j,k]) ≥ (pLatestTxm * y[i]);

/* Restricao (13): para que possa ser        *
 * possivel transmitir uma mensagem k com    *
```

```
 * FID j no ciclo i, deve existir espaco      *
 * disponivel suficiente                      */
s.t. pLatestTx{i in I, j in J, k in K}:
     sum{p in 1..j-1, q in K} (x[i,p,q] * setLF[q])
     + sum{p in 1..j-1} (1 - sum{q in K} x[i,p,q])
     ≤ (gNumberOfMinislots - setLF[k]);

/*           3:Funcao objetivo                */
maximize obj: sum{i in I} y[i];

/*****************************************
 * Conjuntos de Dados                        *
 *****************************************/
data;

param nroMsg := 10;
param setLF :=        /* C */
               1        80
               2        80
  /* Msg */    3        80
               4        80
               5        20
               6        20
               7        20
               8        20
               9        20
              10        20
              ;

param Frame:     /* Mensagem */
   1 2 3 4 5 6 7 8 9 10 :=
1  1 0 0 0 0 0 0 0 0 0
2  0 1 0 0 0 0 0 0 0 0
3  0 0 1 0 0 0 0 0 0 0
4  0 0 0 1 0 0 0 0 0 0
5  0 0 0 0 1 0 0 0 0 0
6  0 0 0 0 0 1 0 0 0 0
7  0 0 0 0 0 0 1 0 0 0
8  0 0 0 0 0 0 0 1 0 0
9  0 0 0 0 0 0 0 0 1 0
10 0 0 0 0 0 0 0 0 0 1;

param gNumberOfMinislots := 200;
param cMsgEmAnalise      :=  20;
param pLatestTxm         := 180;

end;
```

Apêndice B

Implementação do Método Proposto em (Chokshi and Bhaduri, 2010)

```
function  [bO bLeft] = flexrayService(bIu, bIl, k,
                        tDyn, tCycle, lastX)

import ch.ethz.rtc.kernel.*;
if nargin < 4
    error('Too few arguments');
elseif nargin > 6
    error('Too many arguments');
elseif strcmp(class(bIu),'ch.ethz.rtc.kernel.Curve[]')
    beta = bIu;
    if isscalar(bIl)
        kU = bIl;
        kL = kU;
    else
        kU = bIl(1);
```

```
        kL = bIl(2);
    end
    DYN = k;
    Cycle = tDyn;
    T = tCycle;
elseif strcmp(class(bIu),'ch.ethz.rtc.kernel.Curve')
    beta(1) = bIu;
    beta(2) = bIl;

    if isscalar(k)
        kU = k;
        kL = kU;
    else
        kU = k(1);
        kL = k(2);
    end
    DYN = tDyn;
    Cycle = tCycle;
    T = lastX;
end

% Addaption of the Beta curve (Chokshi and Bhaduri, 2010)

% Step 1 (beta u and l)
% -> Beta l
% Nullify de comunications cycles of beta^tl containing
% less than kU minislots. Extract kU minislots from the
% remaining comunication cycles.

% -> Beta u
% Nullify de comunications cycles of beta^ul containing
% less than kL minislots. Extract kL minislots from the
% remaining comunication cycles.

dif = Cycle - DYN;
min=rtccurve([0 1 0]);
bI=rtcclone(beta);
for i =(0:Cycle:T)
    capacity = rtcvalue(beta(1),i+DYN,0)-
           rtcvalue(beta(1),i,0);
    if(capacity<kL)
       beta(1)=rtcminus(beta(1),
       rtctimes(capacity,rtcaffine(min,1.0,i+DYN)));
    else
       beta(1)=rtcminus(beta(1),
       rtctimes(capacity-kL,rtcaffine(min,1.0,i+DYN)));
    end

    i2=i+dif;
```

```
    capacity = rtcvalue(beta(2),i2+DYN,0)-
           rtcvalue(beta(2),i2,0);
    if(capacity<kU)
       beta(2)=rtcminus(beta(2),
       rtctimes(capacity,rtcaffine(min,1.0,i2+DYN)));
    else
       beta(2)=rtcminus(beta(2),
       rtctimes(capacity-kU,rtcaffine(min,1.0,i2+DYN)));
    end
end

beta(1)= CurveMath.maxPlusDeconv0(beta(1));
beta(2)= CurveMath.maxPlusDeconv0(beta(2));
bL(1) = rtcmaxconv(rtcminus(bI(1),beta(1)),0);
bL(2) = rtcmaxconv(rtcminus(bI(2),beta(2)),0);
bLeft = bL;

%% Step 2 (beta u and l)

% For each increasing segment of the curve generated in
% step 1, discretize that segment at the point it
% starts increasing

beta(1) = discretize(beta(1),T);
beta(2) = discretize(beta(2),T);

%% Step 3 (beta l)
%Shift the curve by d times unit to obtain the final curve

beta(2).move(DYN,0);
bO = beta;
```

Bibliografia

AUDSLEY, N.; BURNS, A.; RICHARDSON, M.; TINDELL, K.; WELLINGS, A. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 1993.

AUTOSAR. *Site Oficial do AUTOSAR*. 2010. Disponível em `http://www.autosar.org/`. Último acesso em 26/05/2010.

BERTHOMIEU, B.; DIAZ, M. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 1991.

BLACK, P. E. *Knapsack Problem*. 2010. Disponível em `http://www.itl.nist.gov/div897/sqg/dads/HTML/knapsackProblem.html`. Último acesso em 27/07/2010.

BLACK, P. E. *NP-complete*. 2010. Disponível em `http://www.itl.nist.gov/div897/sqg/dads/HTML/npcomplete.html`. Último acesso em 27/07/2010.

BMW Automobiles. *BMW Technology Guide*. 2010. Disponível em `http://www.bmw.com/com/en/insights/technology/technology_guide/articles/flex_ray.html`. Último acesso em 20/05/2010.

BOUDEC, J. L.; THIRAN, P. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. [S.l.]: Springer Verlag, 2001.

CHAKRABORTY, S.; KüNZLI, S.; THIELE, L. A General Framework for Analysing System Properties in Platform-based Embedded System Designs. In: . [S.l.: s.n.], 2003.

CHOKSHI, D.; BHADURI, P. Performance Analysis of FlexRay-based Systems Using Real-Time Calculus, Revisited. In: *Symposium On Applied Computing*. [S.l.: s.n.], 2010.

CLARKE, E.; GRUMBERG, O.; PELED, D. *Model Checking*. [S.l.]: Springer, 1999.

CONSORTIUM, F. Flexray electrical physical layer specification v2.1 rev.b. 2005.

DAVIS, R.; BURNS, A.; BRIL, R.; LUKKIEN, J. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 2007.

DING, S.; TOMIYAMA, H.; TAKADA, H. An Effective GA-based Scheduling Algorithm for FlexRay Systems. *Ieice Transactions on Information and Systems*, 2008.

EMERSON, E. *Temporal and Modal Logic, Handbook of Theoretical Computer Science: Formal Models and Semantics*. [S.l.]: MIT Press, Cambridge, MA, 1991.

FARBER, P. M.; MERLIN, D. J. Recoverability of Communication Protocols - Implications of a Theoretical Study. *IEEE Trans. Commun.*, 1976.

FIATFORUM. *The FIAT Forum*. 2010. Disponível em `http://www.fiatforum.com/stilo/`. Último acesso em 30/07/2010.

FLEXRAY. *FlexRay Communications System Protocol Specification Version 2.1*. 2005. Disponível em `http://www.flexray.com/`. Último acesso em 15/05/2010.

GARDEY, G.; LIME, D.; MAGNIN, M.; ROUX, O. Romeo: A Tool for Analyzing Time Petri Nets. *Lecture Notes in Computer Science*, 2005.

GLPK (GNU Linear Programming Kit). *Site Oficial do GLPK*. 2010. Disponível em `http://http://www.gnu.org/software/glpk/`. Último acesso em 26/05/2010.

GRENIER, M.; HAVET, L.; NAVET, N. Configuring the Communication on FlexRay - The Case of the Static Segment. *Proceedings of ERTS*, 2008.

GUERMAZI, R.; GEORGE, L. Worst Case End-to-end Response Times of Periodic Tasks With an AUTOSAR/FlexRay Infrastructure. In: . [S.l.: s.n.], 2008. Proceedings of 7 th International Workshop on RealTime Networks (RTN08).

HAGIESCU, A.; BORDOLOI, U.; CHAKRABORTY, S.; SAMPATH, P.; GANESAN, P.; RAMESH, S. Performance Analysis of FlexRay-based ECU Networks. ACM Press New York, NY, USA, 2007.

KESKIN, U. In-Vehicle Communication Networks: a Literature Survey. 2009.

KOPETZ, H.; OBERMAISSER, R. Temporal composability [Real-Time Embedded Systems]. *Computing And Control Engineering Journal*, 2002.

Labbe, M. and Laporte, G. and Martello, S. An Exact Algorithm for the Dual Bin Packing Problem. *Operations Research Letters*, 1995.

LORENZ, T. *Advanced Gateways in Automotive Applications*. Tese (Doutorado) — Elektrotechnik und Informatik der Technische Universität Berlin, 2008.

NASPOLINI, A. C. *Modelagem e Verificação de Escalonabilidade de Sistemas de Tempo Real*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Brazil, 2006.

NAVET, N.; SIMONOT-LION, F. *Automotive Embedded Systems Handbook*. [S.l.]: CRC, 2008.

NAVET, N.; SONG, Y.; SIMONOT-LION, F.; WILWERT, C. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, v. 93, n. 6, p. 1204–1223, 2005.

OSEK VDX. *OSEK VDX Portal*. 2010. Disponível em `http://portal.osek-vdx.org/`. Último acesso em 26/05/2010.

POP, T.; POP, P.; ELES, P.; PENG, Z.; ANDREI, A. Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems*, 2008.

Real-Time Systems Team at L'Institut de Recherche en Communications et Cybernetique de Nantes. *Roméo - a Tool for Time Petri Nets Analysis*. 2010. Disponível em `http://romeo.rts-software.org/`. Último acesso em 26/05/2010.

SCHEDL, A. Goals and Architecture of FlexRay at BMW. 2007. Slides presented at the Vector FlexRay Symposium, March 2007.

SCHMIDT, E.; SCHMIDT, K. Message Scheduling for the FlexRay Protocol: The Dynamic Segment. *IEEE Transactions on Vehicular Technology*, 2008.

SCHMIDT, E.; SCHMIDT, K. Message Scheduling for the FlexRay Protocol: The Static Segment. *IEEE Transactions on Vehicular Technology*, 2008.

SEO, S.; MOON, T.; KIM, J.; KIM, S.; SON, C.; JEON, J.; HWANG, S. A Gateway System for an Automotive System: LIN, CAN, and FlexRay. In: . [S.l.: s.n.], 2008. 6th IEEE International Conference on Industrial Informatics, INDIN.

TSAI, J.; YANG, S.; CHANG, Y. Timing Constraint Petri Nets and Their Application to Schedulability Analysis of Real-time System Specifications. *IEEE Transactions on Software Engineering*, 1995.

WANDELER, E.; MAXIAGUINE, A.; THIELE, L. Performance Analysis of Greedy Shapers in Real-time Systems. In: . [S.l.: s.n.], 2006.

WANDELER, E.; THIELE, L. *Real-Time Calculus (RTC) Toolbox*. 2006. Disponível em `http://www.mpa.ethz.ch/Rtctoolbox/`. Último acesso em 26/05/2010.

ZENG, H.; ZHENG, W.; NATALE, M. D.; GHOSAL, A.; GIUSTO, P.; SANGIOVANNI-VINCENTELLI, A. Scheduling the FlexRay Bus Using Optimization Techniques. In: . [S.l.: s.n.], 2009.

ZURAWSKI, R. *The Industrial Communication Technology Handbook*. [S.l.]: CRC Press, 2005.