PROJETO DE SISTEMAS DISTRIBUÍDOS E DE TEMPO REAL PARA AUTOMAÇÃO

Raimundo José de Araújo Macêdo
Jean-Marie Alexandre Farines
(Organizadores)

EDUFBA

Projeto de Sistemas Distribuídos e de Tempo Real para Automação

Raimundo José de Araújo Macêdo
Jean-Marie Alexandre Farines
(Organizadores)

Projeto de Sistemas Distribuídos e de Tempo Real para Automação

Salvador
EDUFBA
2018

Grafia atualizada conforme o Acordo Ortográfico da Língua Portuguesa de 1990, em vigor no Brasil desde 2009.

Capa
Maria Clara Tarrrafa

Projeto Gráfico e Editoração
dos autores

Revisão
dos autores

Sistema de Bibliotecas - UFBA

P964

Projeto de Sistemas Distribuídos e de Tempo Real para Automação / Raimundo José de Araújo Macêdo e Jean-Marie Alexandre Farines (organizadores) – Salvador: EDUFBA, 2018.
250 p.: il.; 16,5 x 24,0 cm

ISBN 978-85-232-1675-7

1. Sistemas Distribuídos 2. Automação 3. Mecatrônica I. Macêdo, Raimundo José de Araújo. II. Farines, Jean-Marie Alexandre

CDU 004.75
CDD 005.736

Editora filiada a

ASOCIACION DE EDITORIALES UNIVERSITARIAS DE AMERICA LATINA Y EL CARIBE

Associação Brasileira das Editoras Universitárias

CBaL Câmara Baiana do Livro

EDUFBA
Rua Barão de Jeremoabo, s/n, Campus de Ondina,
40170-115, Salvador-BA, Brasil
Tel/fax: (71) 3283-6164
www.edufba.ufba.br | edufba@ufba.br

Sumário

Apresentação

Sistemas modernos de automação industrial se baseiam fortemente em tecnologias digitais, onde mecanismos de sensoriamento, controle e atuação, estão geralmente dispersos geograficamente e se comunicam através de uma rede de comunicação.

Os benefícios da descentralização são evidentes: facilidade de crescimento incremental e de manutenção, melhor compartilhamento de recursos, possibilidade de maior redundância que viabiliza técnicas de tolerância a falhas, entres outros. Além disso, o mundo moderno incorpora sistemas embarcados em equipamentos convencionais, a exemplo de aeronaves tripuladas ou não, automóveis, robôs moveis, telefones celulares, máquinas de lavar, caracterizando aplicações onde computação convencional e controle estão integrados em um mesmo ambiente distribuído, e muitas vezes, em simbiose com o processo físico a controlar. Esses últimos sistemas são frequentemente chamados de Sistemas Ciber-Físicos (*Cyber-Physical Systems*). Sistemas Ciber-Físicos, a exemplo de casas inteligências, sistemas inteligentes de geração e distribuição de energia (*smart grids*) e redes veiculares móveis (*VANETs*), podem também ser considerados sistemas de automação, pois possuem elementos e características muito similares aos de plantas industriais operando sobre redes de computadores.

Apesar de todas as vantagens, tais sistemas modernos de automação trazem novos desafios. A dispersão de equipamentos e a comunicação sobre redes de computadores dificulta a necessária coordenação e sincronização de ações entre os vários elementos envolvidos no processo de computação e controle. Por exemplo, dados captados por sensores podem sofrer atrasos inesperados na rede que comprometem a qualidade do controle, ou a falha de algum componente computacional pode requerer a reconfiguração imediata do sistema distribuído.

Este livro aborda vários desses desafios e resulta do trabalho cooperativo de investigação científica entre equipes dos programas de pós-graduação em Mecatrônica (PPGM) da Universidade Federal da Bahia (UFBA) e o de Engenharia de Automação e Sistemas (PPGEAS) da Universidade Federal de Santa Catarina (UFSC) – com financiamento da CAPES a partir da Chamada Pública de 2006 do Programa Nacional de Cooperação Acadêmica (PROCAD).

Cada capítulo trata de um desafio importante dos sistemas modernos de automação e é autocontido, portanto podendo ser lido em qualquer ordem. No entanto, recomendamos a leitura do livro na sequência proposta, iniciando pelo capítulo I que apresenta os principais problemas do controle realimentado sobre redes de computadores (*Networ-*

ked Control Systems) e as técnicas utilizadas para resolvê-los. O capítulo II apresenta os principais conceitos de confiabilidade, com foco na detecção de defeitos, e o capítulo III, o projeto de detectores de defeitos voltados para o ambiente distribuído e de automação. Estes dois capítulos devem ser lidos em sequencia. O capítulo IV discute desafios e soluções para a reconfiguração dinâmica de sistemas de tempo real, característicos da automação. Ênfase é dada à reconfiguração de escalonamento de modo a minimizar perdas de prazos (*deadline*) das tarefas típicas de ambientes de automação. O capítulo V apresenta os principais problemas ligados às redes de sensores sem fio, uma tecnologia amplamente utilizada nos dias de hoje para Sistemas Ciber-Físicos. Por fim, o capítulo VI trata dos métodos e das técnicas relativos ao desenvolvimento de sistemas de controle e automação.

Finalmente, gostaríamos agradecer aos colegas e pós-graduandos envolvidos no projeto e na confecção de cada capítulo. Um agradecimento especial segue para Alirio Sá, participante do projeto como estudante e agora como colega professor da UFBA, que nos ajudou no trabalho de formatação dos capítulos através do software LATEX.

Raimundo José de Araújo Macêdo, UFBA

Jean-Marie Farines, UFSC

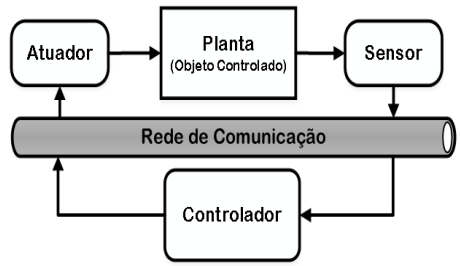
CAPÍTULO 1

Sistemas de Controle via Rede

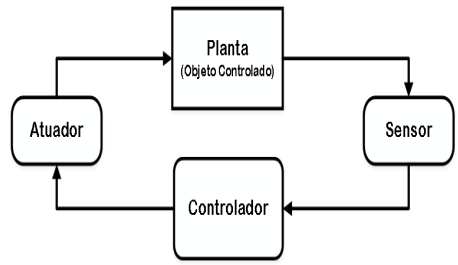
Alirio S. de Sá, Tanisia Foletto, Ubirajara Moreno e Raimundo J. de A. Macêdo

O avanço tecnológico dos sistemas embarcados e das redes de computadores propiciou o desenvolvimento de novas abordagens e arquiteturas computacionais para o controle de plantas industriais cada vez maiores e mais complexas, nas quais os componentes dos sistemas de controle se comunicam e coordenam suas ações usando uma rede de comunicação. Essas novas abordagens facilitam a interoperabilidade entre componentes de fabricantes distintos, diminuem a complexidade da interconexão entre componentes, permitem um monitoramento e supervisão remotos mais eficientes e diminuem os custos operacionais (LIAN; MOYNE; TILBURY, 2001). Essas plantas modernas, no entanto, devem trabalhar de forma coordenada e integrada, o que evidencia e aumenta a criticalidade de aspectos como correção na execução de operações distribuídas, atendimento a restrições temporais das operações e necessidade de tolerar falhas de componentes.

Dentro deste contexto, a expressão *Sistema de Controle via Rede* (**NCS**, *Networked Control System*) foi inicialmente utilizado para designar sistemas de controle onde uma rede de comunicação é usada para interconectar os dispositivos do sistema: sensores, controladores e atuadores (ver Figura 1.1(a)). A utilização de NCSs vem crescendo em aplicações industriais. Quando comparados a sistemas de controle tradicionais (ver Figura 1.1(b)), em que os nós são ligados ponto-a-ponto por meio de cabos dedicados, NCSs são mais fáceis de reconfigurar fisicamente, são mais baratos de implementar e manter e, principalmente, permitem um melhor gerenciamento das informações do sistema, uma vez que os dispositivos podem trocar dados diretamente.



(a) Exemplo de sistema de controle via rede (NCS)



(b) Exemplo de sistema de controle tradicional

Figura 1.1: Exemplos de sistemas de controle via rede e tradicional

Nas últimas décadas, as redes de comunicação com aplicações na indústria foram uma das tecnologias que mais evoluíram na área de controle industrial (MOYNE; TILBURY, 2007). Sistemas de controle com aplicação industrial requerem redes de comunicação com características de tempo real – isto é, que transmitam os dados de forma confiável, dentro de prazos específicos, evitando degradação de desempenho ou ainda perda de estabilidade do sistema de controle. Inicialmente, as redes utilizadas para a implementação de malhas de controle eram cabeadas, entretanto nos últimos anos redes sem fio têm se mostrado uma importante alternativa, seja por particularidades do processo controlado ou planta, pela facilidade de reconfiguração da estrutura física, ou pela utilização na construção de redes de sensores.

A forma como os nós (sensores, atuadores e controladores) acessam a rede é determinada por um protocolo de controle de acesso ao meio ou MAC (*Medium Acces Control*). Diversos protocolos de comunicação que atendem a requisitos de tempo real em NCS foram desenvolvidos, tais como: CAN, Ethernet (em vários perfis de tempo real), DeviceNet, PROFIBUS, IEEE 802.11e e Bluetooth (IRWIN; COLANDAIRAJ; SCANLON, 2006; LIAN; MOYNE; TILBURY, 2001; HE; GUO, 2008; PLOPLYS; KAWKA; ALLEYNE, 2004). Protocolos de acesso ao meio são de extrema impor-

tância na realização do projeto do NCS, uma vez que são responsáveis por uma parte relevante do atraso de comunicação e da sua regularidade. Assim, dependendo do tipo de aplicação e dos respectivos requisitos temporais, alguns protocolos são mais adequados que outros.

Mais recentemente, o termo NCS também passou a abranger uma outra classe de problema em que controladores distribuídos colaboram e coordenam suas ações a partir da rede de comunicação para realizar tarefas individuais e coletivas. Na literatura, estes sistemas são por vezes designados como sistemas de controle cooperativos (SHAMMA; ARSLAN, 2007). Como exemplos destas aplicações, destacam-se sistemas compostos por robôs móveis que realizam tarefas tais como perseguição e evasão, localização, busca e resgate, jogos de equipe como futebol de robôs, entre outras. Sistemas de controle cooperativos, considerando as diversas aplicações, herdam várias questões subjacentes comuns aos sistemas de controle via rede – tanto do ponto de vista das restrições como da integração dos diversos elementos que compõe estes sistemas.

Em geral, os problemas básicos em sistemas de controle via rede estão relacionados aos aspectos do controle e às questões de rede e tempo real. No caso do controle, a estrutura básica é constituída por controladores, que são responsáveis pela execução das tarefas atendendo requisitos de desempenho da aplicação e estabilidade, por sensores que são responsáveis pela aquisição e transmissão de dados em taxas que permitam a execução das tarefas, e pelos atuadores que agem diretamento sobre o processo. As ações de cada um desses elementos devem atender a requisitos temporais e de garantia de entrega de informações via rede. Na perspectiva do controle, as questões de tempo real e rede acabam sendo consideradas como restrições, que limitam a taxa de amostragem dos sensores, taxa de atualização do cálculo da lei de controle e atuação. No caso de sistemas embarcados, as restrições de processamento acabam por restringir a complexidade do controlador.

Tradicionalmente, a concepção de sistemas de controle via rede (e sistemas de controle cooperativos) envolve diversas áreas de conhecimento e vem sendo tratada por comunidades distintas, tais como a comunidade de controle de sistemas, comunidade de tempo real e, mais recentemente, a comunidade que trata de sistemas robóticos cooperativos (SHAMMA; ARSLAN, 2007).

No caso das comunidades de controle automático e robótica, as questões de tempo real e rede, como observado anteriormente, aparecem geralmente apenas como restrições ao problema. O objetivo torna-se projetar um controlador que garanta que a estabilidade do sistema realimentado seja tolerante a variações de atraso e perda de mensagens e, possivelmente, que minimize a troca de informações pela rede, sem no entanto propor alterações nas políticas de acesso ao meio e de escalonamento das transmissões,

consumo de energia e de restrições de processamento entre outros (LEMMON, 2010; BAILLIEUL; ANTSAKLIS, 2007). De outra parte, a comunidade de tempo real busca propor alterações nestas políticas de modo a melhorar o desempenho da rede e reduzir o consumo de energia, como é o caso de redes de sensores, sem no entanto verificar o impacto destas estratégias no desempenho do sistema controlado.

Apesar de grande parte dos trabalhos seguirem esta separação, algumas técnicas e formulações têm sido propostas para se obter um projeto integrado, que leve em consideração tanto as necessidades de desempenho do controle como o desempenho da estrutura de comunicação e processamento. Inicialmente estas estratégias foram designadas na literatura como técnicas de *co-design* (TORNGREN et al., 2006). Esta abordagem integrada se faz necessária para que o sistema, como um todo, tenha seu desempenho melhorado, e que o comportamento deste tipo de sistema seja melhor compreendido. Mais recentemente, novas estratégias de controle têm sido propostas para incorporar um melhor desempenho da rede, como é o caso, por exemplo, dos sistemas de controle baseados em evento (*self-triggered control systems*) (LOZOYA; MARTÍ; VELASCO, 2010; ANTA; TABUADA, 2010; ARAÚJO et al., 2011; PARK; ARAÚJO; JOHANSSON, 2011).

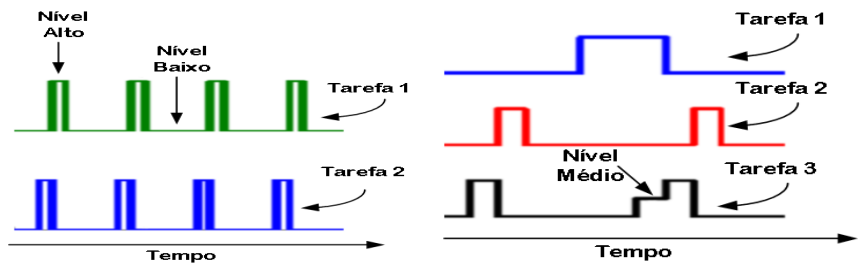
Fundamentalmente, a concepção de um sistema de controle via rede traz uma série de desafios de projeto, relacionados principalmente à manutenção da qualidade do controle realizado diante das particularidades da execução deste tipo de sistema em uma infraestrutura de rede de comunicação, como por exemplo: perda de mensagens; atrasos de comunicação e de processamento variados; escalonamento das tarefas nos dispositivos do sistema etc.

Nas próximas seções são discutidos alguns aspectos básicos relacionados à análise e ao projeto dos Sistemas de Controle via rede. Para isto, primeiramente são apresentados alguns desafios relacionados aos sistemas de controle via rede e, em seguida, são apontados alguns marcos da pesquisa em controle via rede. Por fim, são analisados aspectos relacionados aos impactos da rede de comunicação no desempenho do controle realizado – o que inclui discussões do sistema de controle via rede sujeito a atrasos variados e perda de mensagens.

1.1 Desafios em Sistemas de Controle Via Rede

A inserção de uma rede de comunicação em uma malha de controle torna a análise e o projeto do sistema de controle mais complexos, pois aspectos técnicos relacionados a diferentes áreas de conhecimento, como sistemas de controle, sistemas de tempo real,

redes de comunicação e sistemas distribuídos, precisam ser levados em consideração (BAILLIEUL; ANTSAKLIS, 2007). Em sistemas de controle via rede, as tarefas e/ou mensagens de cada dispositivo conectado a rede de comunicação precisam ser escalonadas corretamente, de forma a definir qual destes dispositivos terá acesso ao meio de comunicação em um determinado instante (STEMMER, 2010).



(a) Escalonamento de duas tarefas

(b) Escalonamento de três tarefas

Figura 1.2: Escalonamento de tarefas em uma rede de comunicação

A Figura 1.2 apresenta exemplos com escalonamento de tarefas em dispositivos que compartilham um mesmo meio de comunicação para transmissão de mensagens. Na Figura 1.2(a), duas tarefas são escalonadas de tal forma que não há disputas pelo uso do meio de comunicação para transmissão de mensagens, isto é indicado no gráfico da figura da seguinte forma: o dispositivo está transmitindo (nível alto) ou o dispositivo não possui transmissões a serem realizadas (nível baixo). Neste exemplo, é possível observar que o intervalo entre as transmissões se mantêm constantes e periódicos. Com isso, é possível escalonar todas as tarefas, de modo que se cumpram os prazos corretamente, com atrasos determinísticos na entrega das mensagens. No cenário com três tarefas apresentado na Figura 1.2(b), as transmissões não são regulares e ocorrem níveis médios, indicando que um nó está aguardando a liberação do meio de comunicação para tentar enviar uma mensagem. Essas irregularidades nas transmissões das mensagens causam atrasos aleatórios, denominados por *jitter* de comunicação. Em alguns casos práticos, quando o atraso é maior que um período de amostragem a mensagem é considerada como perdida.

Os efeitos do *jitter* de comunicação e das perdas de mensagens são alguns dos principais desafios do projeto de sistemas de controle via rede, pois podem afetar diretamente o desempenho do sistema ou ainda desestabilizá-lo. Outros efeitos também podem ser inseridos pelas redes de transmissão, como por exemplo: o efeito da quantização, que é característico de sistemas digitais; e a restrição na escolha do período de

amostragem, que se deve, por exemplo, à limitação na capacidade de transmissão da rede de comunicação (i.e., limitação na largura de banda da rede).

1.1.1 Atrasos e Perdas de Mensagens em NCS

Em um laço de controle, o atraso de atuação (R_{loop}) representa o intervalo de tempo entre a amostragem e a execução da ação de controle sobre a planta. Em um sistema de controle via rede, o atraso de atuação inclui os atrasos de processamento e de comunicação (Figura 1.3).

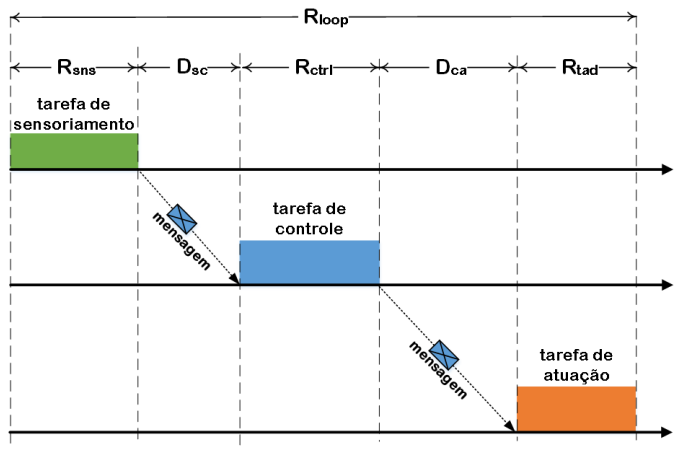


Figura 1.3: Atrasos em Controle via Rede

Para um NCS, o atraso de atuação (R_{loop}) pode ser calculado da seguinte forma:

$$R_{loop} = R_{sns} + D_{sc} + R_{ctrl} + D_{ca} + R_{atd} \tag{1.1}$$

em que: R_{sns}, R_{ctrl} e R_{atd} representam, respectivamente, os tempos de respostas da tarefa de sensoriamento que executa no sensor, da tarefa de controle que executa no controlador e da tarefa de atuação que executa no atuador (ver Figura 1.3); e D_{sc} e D_{ca} representam os atrasos de comunicação entre sensor e controlador e entre controlador e atuador, respectivamente (ver Figura 1.3).

As parcelas de atraso R_{sns}, R_{ctrl} e R_{atd} dependem dos diferentes elementos que impactam nos atrasos de processamento nos respectivos dispositivo. Entretanto, quando as tarefas executam sobre sistemas operacionais de tempo real, estes atrasos podem ser obtidos através de estratégias tradicionais de análise de tempo de resposta em sistemas de tempo real (CHENG, 2002) – desta forma, estes atrasos podem, em muitos casos, ser

classificados como determinísticos, sendo possível, por exemplo, realizar estimativas das magnitudes destes atrasos no melhor e no pior caso. As diferenças entre os atrasos de processamento no pior e no melhor caso implicam em um *jitter* de processamento que tem influência no desempenho do controle (CERVIN et al., 2004).

As parcelas de atraso D_{sc} e D_{ca} dependem das características associadas à rede de comunicação, tais como protocolo de acesso ao meio, taxas de transmissão, prioridade de encaminhamento de mensagens, entre outros – o que pode implicar em atrasos variados (*jitter* de comunicação) e, em alguns casos, não determinísticos.

O *jitter* de comunicação, em muitos casos, são mais nocivos ao desempenho do controle que o *jitter* de processamento, por conta da menor previsibilidade e das magnitudes das variações dos atrasos de comunicação. Em NCS, os efeitos combinados dos *jitter* de processamento e de comunicação é denominado de *jitter* de controle, uma vez que provocam variações nos intervalos entre as ações do controlador sobre a planta.

Perdas de mensagens ocorrem em função de fatores como: colisão de mensagens; erros nas transmissões das mensagens; congestionamento na rede etc. Em aplicações de controle, as perdas de mensagens, assim como o *jitter*, degradam o desempenho e também podem desestabilizar o sistema. Normalmente em NCS, a planta ainda tolera uma determinada taxa de perda de mensagens sem desestabilizar o sistema, mas nem sempre essas taxas são fáceis de serem determinadas, pois dependerá de cada tipo de processo controlado.

Os impactos da rede de comunicação sobre o sistema de controle são demonstrados com exemplos na próxima Seção.

1.1.2 Efeitos da Rede no Sistema de Controle: Exemplo Motivador

Para ilustrar os efeitos da rede de comunicação nos sistemas de controle, realizamos algumas simulações usando o simulador*TrueTime*, que é um pacote implementado no Matlab/Simulink (HENRIKSSON; CERVIN; ÅRZÉN, 2002).

Incialmente foi simulado um sistema de controle tradicional com controlador digital ideal. Em seguida, uma rede de comunicação foi introduzida para interconectar controlador, sensor e atuador. Assim como em Cervin, Ohlin e Henriksson (2007), a planta simulada representa um motor servo DC, descrito pela função de transferência da Equação 1.2.

$$G(s) = \frac{1000}{s(s+1)} \tag{1.2}$$

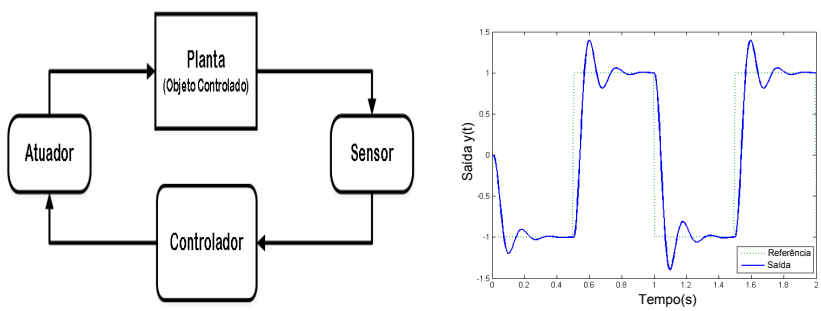
O sistema deve controlar a posição angular do eixo do motor DC a partir de uma

entrada de tensão. O controle é realizado por um controlador Proporcional-Integral-Derivativo (PID) discreto com parâmetros definidos conforme proposto em Cervin, Ohlin e Henriksson (2007), isto é: Ganho proporcional $K_p = 1.5$; ganho integral $T_i = 0.12$; e ganho derivativo $T_d = 0.049$.

Além disso, foram considerados três cenários para avaliação da influência da rede de comunicação sobre o desempenho do sistema de controle: (i) Sistema de controle digital tradicional (sem rede de comunicação); (ii) Sistema de controle via rede, implementado usando rede cabeada; e (iii) Sistema de controle via rede, implementado usando rede sem fio. Esses três cenários utilizam um mesmo modelo de ativação de tarefas, no qual a tarefa de sensoriamento é periódica, sendo acionada a cada período de amostragem de 12 milissegundos; a tarefa de controle é dirigida a evento e é acionada quando uma amostra é recebida pelo controlador; e a tarefa de atuação é dirigida a evento e é acionada quando uma ação de controle é recebida pelo atuador.

Cenário 1: Sistema de Controle Digital Tradicional

Este cenário é usado para evidenciar que com o controlador diretamente conectado aos dispositivos sensor e atuador (Figura 1.4(a)) e, na ausência de perturbações externas, o sistema garante que a resposta do motor-DC segue a referência (Figura 1.4(b)).



(a) Sistema de controle tradicional

(b) Resposta do sistema de controle

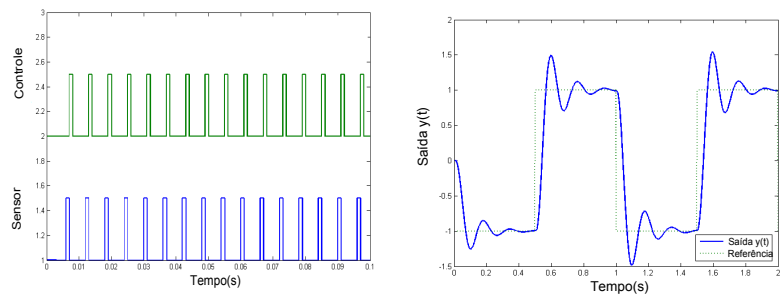
Figura 1.4: Avaliação de resposta de sistema de controle tradicional

Cenários 2 e 3: Sistema de Controle via Rede

Para análise dos efeitos da rede de comunicação no desempenho do controle, dois tipos de meios de comunicação são considerados: (I) rede cabeada com protocolo CAN (Controller Area Network); e (II) rede sem fio com o protocolo IEEE 802.11b.

Cenário 2: NCS com rede CAN. Neste cenário, o NCS usa uma CAN com taxa de transmissão de $1Mbps$ e política de priorização de mensagens. Na rede, as mensagens enviadas pelo controlador são as mais prioritárias.

- *NCS/CAN sem disputa no acesso ao meio de comunicação (Figura 1.5).* Neste caso, quando se observa o escalonamento das mensagens na rede (Figura 1.5(a)) e a saída do sistema (Figura 1.5(b)), é possível notar que não existem atrasos variádos nas transmissões das mensagens e, portanto, o desempenho do controle é pouco afetado.



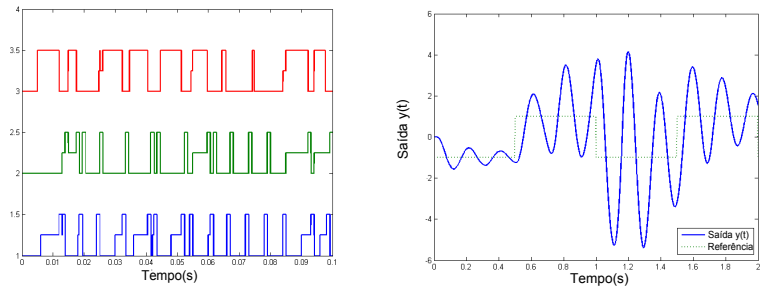
(a) Escalonamento da Rede (b) Saída do Sistema

Figura 1.5: Desempenho do NCS/CAN livre de disputa por acesso ao meio

- *NCS/CAN com disputa no acesso ao meio de comunicação (Figura 1.6).* Para demonstrar os efeitos do *jitter* de comunicação no desempenho do controle, um novo dispositivo é adicionado na rede para gerar interferência na comunicação entre os dispositivos do sistema de controle. Esse novo dispositivo envia mensagens a cada 9.9 milissegundos, de modo a provocar disputa por acesso ao meio de comunicação. Essa disputa entre dispositivos provoca atrasos na entrega das mensagens (Figura 1.6(a)), prejudicando o desempenho do sistema (Figura 1.6(b)).

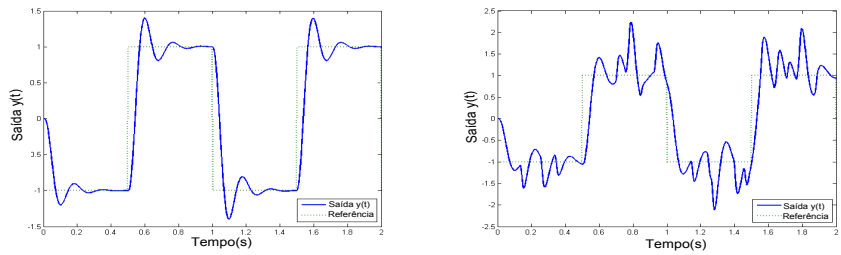
Cenário 3: NCS com rede fio. Neste cenário, o NCS usa uma rede sem fio com protocolo IEEE 802.11b e taxa de transmissão de 52 Mbps. Quando não há jitter de comunicação ou perda de mensagens, o desempenho do sistema é pouco degrado (Figura 1.7(a)). Ao simular uma taxa de 20% de perda de mensagens na rede, há uma degradação acentuada no desempenho do controle (Figura 1.7(b)).

No projeto de um sistema de controle, um requisito básico é a estabilidade. A presença de *jitter* de comunicação e perdas de mensagens pode degradar o desempenho do



(a) Escalonamento da Rede (b) Saída do Sistema

Figura 1.6: Desempenho do NCS/CAN com disputa por acesso ao meio



(a) Saída do Sistema sem perda de mensagens (b) Saída do Sistema com perda de mensagens

Figura 1.7: Desempenho do NCS com rede 802.11b

NCS ou até mesmo desestabilizá-lo. Portanto, ressalta-se a importância de realizar um projeto adequado para os sistemas de controle via rede, de modo a atender aos requisitos de tempo real impostos pela aplicação de controle, face aos efeitos degenerativos da rede de comunicação.

1.2 Histórico dos Sistemas de Controle via Rede

A Figura 1.8 ilustra a linha do tempo da evolução da teoria de controle clássica até sistemas de controle via rede. Sistemas de controle com componentes espacialmente distribuídos têm existido por décadas, com uma teoria que já se encontra bem consolidada. Os sistemas de controle via rede surgiram por volta dos anos 80 e hoje se apresentam como uma linha de pesquisa promissora em sistemas dinâmicos.

Halevi e Ray (1988), Luck e Ray (1990) e Hong (1995) representam os primeiros estudos que tratam de análise e síntese de controladores com aplicação de uma rede

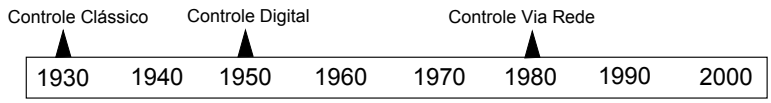


Figura 1.8: Linha do tempo evidenciando alguns marcos das pesquisas em sistemas de controle (BAILLIEUL; ANTSAKLIS, 2007)

de comunicação. Halevi e Ray (1988) propuseram uma metodologia para controlar uma planta sobre uma rede periódica com atrasos. Luck e Ray (1990) desenvolveram o estudo baseados na metodologia das filas, que usa um observador de estados para estimar os estados da planta física e um preditor para computar o controle preditivo baseado sob a medição das saídas passadas. Hong (1995) desenvolveu uma metodologia de escalonamentos do intervalo de amostragem para um NCS de forma que os atrasos na rede não afetem significativamente o desempenho e a estabilidade do sistema de controle.

Os trabalhos anteriores são precursores em sistemas de controle via rede, no entanto, o tema só passa a ganhar atenção na comunidade de sistemas de controle a partir dos resultados apresentados por Nilsson (1998), o qual apresentou uma metodologia de controle estocástico ótimo para controlar um NCS sob uma rede com atrasos aleatórios. Para isto, Nilsson (1998) considera que os atrasos variáveis são menores que o período de amostragem, essa premissa ainda é utilizada em muitos trabalhos envolvendo NCS, até os dias de hoje.

Outro marco importante nas pesquisas de NCS, foi o trabalho de Murray e outros (2003), o qual discutiu as novas tecnologias que vêm surgindo e as necessidades de enquadramento de novas pesquisas para aplicações que não envolvem só questões de controle, mas também relativas a rede de comunicação. A partir disso, os *Sistemas de Controle via Rede* passam a ser uma nova e promissora área de pesquisa.

A inclusão de uma rede de comunicação em sistema de controle exige mudanças na modelagem, análise e projeto, pois introduz diversas incertezas ao sistema de controle. Atualmente a literatura é muito ampla e não se restringe a um único tema, já que se trata de um assunto multidisciplinar. As pesquisas têm sido realizadas sobre diversos aspectos e vários artigos trazem uma visão geral sobre o tema *Sistema de Controle via Rede*, por exemplo, ver Zhang, Branicky e Phillips (2001), Baillieul e Antsaklis (2007), Hespanha, Naghshtabrizi e Xu (2007) e Tang e Yu (2007).

As pesquisas em NCS apresentam diferentes aspectos na análise e projeto e podem ser divididas nas seguintes categorias: análise de estabilidade; síntese de controladores e *co-desing*; NCS não lineares; e estimação etc. Uma noção geral das pesquisas em cada categoria é apresentada a seguir.

1.2.1 Garantias de Estabilidade

A maior parte dos estudos envolvendo análise de estabilidade de NCS trata de questões relacionadas a atrasos variáveis ou perdas de mensagens. Em geral, as abordagens de análise de estabilidade são classificadas em dois tipos: determinísticas e estocásticas (SEILER; SENGUPTA, 2005).

Nas abordagens determinísticas, destaca-se o uso da teoria de Lyapunov para fornecer garantias de estabilidade. Naghshtabrizi e Hespanha (2005) e Jiang e Han (2006), por exemplo, usam teoria de Lyaponov para fornecer garantias de estabilidade para sistemas com atrasos variantes no tempo. Outro exemplo pode ser encontrado em Zhang, Branicky e Phillips (2001) e em Elia (2005), os quais utilizam a teoria de Lyapunov para determinar a máxima taxa de perda de mensagens que pode ser tolerada sem que o sistema perca a estabilidade.

Dentre as abordagens estocásticas para análise de estabilidade em sistemas de controle via rede, o trabalho de Nilsson, Bernhardsson e Wittenmark (1998) se destacam, pois propõem um teste para a estabilidade de sistemas estocásticos sujeitos a atrasos aleatórios. Os trabalhos de Xie e Xie (2009) e de Xiong e Lam (2007) também se destacam, pois analisam a estabilidade estocástica de sistemas de controle via rede com perdas aleatórias de mensagens.

Desigualdades matriciais (LMI, *Linear Matrix Inequality*) é outra técnica utilizada para a análise de estabilidade em NCS, com perdas de mensagens e atrasos. Exemplos do uso desta técnica podem ser encontrados em Yu e outros (2005) e em Naghshtabrizi e Hespanha (2006).

Kao e Lincoln (2004) apresentam uma análise da estabilidade utilizando métodos frequenciais, para Sistemas de Controle via Rede com atrasos variantes no tempo. A análise da influência da taxa de amostragem na estabilidade do sistema, determinando o máximo atraso que fornece garantias de estabilidade ao sistema, é estudada nos trabalhos de Branicky, Phillips e Zhang (2000) e Santos, Moreno e Montez (2007a).

1.2.2 Síntese de Controladores e *co-desing*

Várias técnicas de controle vêm sendo empregadas em NCS para melhorar o desempenho e a estabilidade desses sistemas, dependendo do problema que se está tratando. Estruturas preditivas e adaptativas aplicadas em NCS são mostradas, por exemplo, nos trabalhos de Beldiman, Bushnell e Walsh (2000); de Jing, Liqian e Tongwen (2007); e de Zhao, Liu e Rees (2008).

Métodos de controle H_∞ têm sido propostos para perda de mensagens[1], para atrasos[2] e efeito da quantização[3]. Técnicas de controle estocástico com atrasos aleatórios têm sido empregadas em NCSs para considerar o *jitter* de comunicação[4] e perda de mensagens[5].

Um dos maiores desafios do projeto de NCS é manter o desempenho do controle em ambientes multi-tarefa e de tempo real, nos quais também precisam ser tratadas as questões relacionadas ao escalonamento de tarefas nos dispositivos e das mensagens na rede. Como visto anteriormente, projetos que levam em conta estes aspectos são denominados projetos de *co-desing*. Dentre estes, é possível citar os trabalhos de Cervin e outros (2004); Santos, Moreno e Montez (2007b); e Xia, Sun e Tian (2009). Para análise de desempenho, destacam-se as técnicas Margem de *Jitter*, proposta por Cervin e outros (2004), e de Qualidade de Controle, proposta por Martí e outros (2004).

1.2.3 Sistemas Não Lineares de Controle Via Rede

Embora já tenha se desenvolvido muitos trabalhos em sistemas de controle via rede, abordando diversos tópicos, a sua maioria é voltada para sistemas lineares. A modelagem de Sistemas Não Lineares de Controle Via Rede (NCSs Não Lineares) tem despertado o interesse da comunidade científica, pois não linearidades geralmente existem em problemas práticos dos sistemas de controle.

Resultados relevantes sobre estabilidade para NCSs não lineares, têm sido publicados. Dentre eles, o trabalho de Walsh, Beldiman e Bushnell (1999) é um dos precursores da análise de estabilidade de sistemas não lineares de controle via rede. Neste trabalho, os autores utilizam metodologias de controle e um protocolo que fornece garantias de estabilidade para um NCS com limitação de banda.

Cao, Zhong e Hu (2008) propõem condições de estabilidade para NCS dependente de atrasos e perturbações não lineares. Essas condições são baseadas na teoria de Lyapunov combinadas com técnicas de modelagem. O trabalho de Polushin, Liu e Lung (2008) trata do problema de estabilização de um NCS não linear baseado em modelo discreto, sujeitos a restrições impostas pela rede como atrasos, perdas de mensagens e variação no período de amostragem. Um estudo da estabilidade de um NCS não linear com perdas de mensagem pode ser encontrado no trabalho de Mastellone, Dorato e Abdallah (2006).

[1] ver, por exemplo, Ishii (2008)
[2] ver, por exemplo, Yang e outros (2006)
[3] ver, por exemplo, Peng e Tian (2007), Gao, Chen e Lam (2008) e Yue, Han e Lam (2005)
[4] ver, por exemplo, Nilsson, Bernhardsson e Wittenmark (1998)
[5] ver, por exemplo, Gupta, Hassibi e Murray (2007)

1.2.4 Estimação para NCS

O problema de estimação de estado, no contexto dos sistemas de controle, consiste em usar amostras passadas da saída da planta, em conjunto com algum conhecimento prévio de sua dinâmica e dos dispositivos envolvidos no laço de controle, para produzir estimativas sequenciais sobre o estado do sistema que, por exemplo, pode apenas ser observado indiretamente ou sob condições que induzem incertezas sobre estes estados.

Em NCS, o problema da estimação de estado tem ganhado atenção considerável, principalmente quando o sistema é implementado via redes sem fio, nas quais o desempenho do sistema é deteriorado principalmente devido a perdas de mensagens e aos efeitos da quantização. Neste contexto, utilizar um observador como estimador permitirá que as redes sem fio ganhem cada vez mais espaço em aplicações industriais, pois na ocorrência de perdas de mensagens, mesmo que estimada, a medida será entregue para o controlador.

Estimadores baseados em filtragem de Kalman têm se destacado em muitos trabalhos em NCSs com perdas de mensagens e atrasos, tais como em Gupta, Hassibi e Murray (2007); Xu e Hespanha (2005); Sinopoli e outros (2004); Ray, Liou e Shen (1993); Wang, Ho e Liu (2004); e Gupta e outros (2006).

O problema de estimação não linear para NCS com perda de mensagens tem sido pouco investigado na literatura. O desempenho de um NCS com perdas de mensagens, utilizando o Filtro de Kalman Estendido e o um filtro baseado em estimação de estados com horizonte móvel (MHE), é analisado quando esses filtros são utilizados como estimadores. Muraca, Pugliese e Rocca (2008), por exemplo, analisam o desempenho em uma rede de sensores sem fio, quando o Filtro de Kalman Estendido é utilizado com estimador.

1.3 Análise de Desempenho de Sistema de Controle via Rede sujeito a Atrasos Variados

Nesta Seção, é apresentado um procedimento para avaliar a Qualidade do Controle (QoC, *Quality of Control*) em NCSs com atrasos aleatórios (*jitter*). Para isto, o efeito do atraso é quantificado por meio de uma métrica conhecida como margem de fase aparente (CERVIN et al., 2004).

A abordagem analítica, usando a margem de fase aparente combinada com o uso de alguns conceitos de teoria das filas, permite estabelecer os limites de uso da rede de comunicação, definindo, por exemplo, o número de plantas que podem ser controladas

via rede e a degradação de desempenho dos sistemas de controle das mesmas.

A margem de fase aparente é utilizada como métrica de avaliação da QoC do sistema (MARTÍ et al., 2004), pois permite avaliar tanto questões de robustez quanto de desempenho num único índice.

Uma visão mais detalhada do procedimento de análise e dos conceitos associados são apresentados nas subseções a seguir.

1.3.1 Margens de Jitter e de Fase Aparente

A *margem de atraso* (L_m), a *margem de ganho* (g_m) e a *margem de fase* (φ_m) são conceitos da teoria de controle clássico que expressam o grau de incerteza que um dado sistema nominal suporta sem que este se torne instável. Estes conceitos são expressos sob a forma de métricas em termos de variação de atraso, ganho ou fase de um dado processo linear (FRANKLIN; POWELL; EMAMI-NAEINI, 2002). A *margem de fase* e a *margem de atraso* foram concebidas dentro de um universo de sistemas com incertezas paramétricas sujeitos a atrasos constantes (OGATA, 2009).

Kao e Lincoln (2004) propuseram um critério de estabilidade para sistemas que possuem atrasos variáveis – usando um modelo similar ao exemplificado na Figura 1.9, na qual L representa uma parcela de atraso constante e J representa uma parcela de atraso variável (*jitter*).

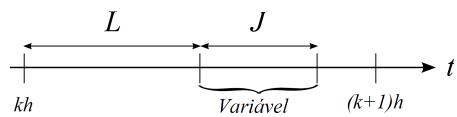


Figura 1.9: Atraso variável formado por uma parte mínima L e uma variável J

No trabalho de Cervin e outros (2004), as noções de *margem de jitter* e *margem de fase aparente* são introduzidas na tentativa de estender as noções clássicas para sistemas sujeitos a atrasos que variam no tempo. A *margem de jitter* está fundamentada no critério de estabilidade proposto por Kao e Lincoln (2004) e cuja versão de Cervin e outros (2004) é reproduzida a seguir:

Teorema 1.3.1 (Estabilidade sob jitter) *O sistema de malha fechada da Figura 1.10 é estável para qualquer atraso variante no tempo $\Delta \in [0, N \cdot h]$, em que $N > 0$ é um número real e h é o período de amostragem, se*

$$\left| \frac{P_{alias}(\omega)K(e^{j\omega})}{1 + P_{ZOH}(e^{j\omega})K(e^{j\omega})} \right| < \frac{1}{\tilde{N}|e^{j\omega} - 1|}, \forall\ \omega \in [0, \pi], \tag{1.3}$$

onde $\tilde{N} = \sqrt{\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g}$, g *é a parte fracionária de* N ,$g = N - \lfloor N \rfloor$, $P_{ZOH}(z)$ *é a discretização com o sustentador de ordem zero de* $P(s)$ *e* $P_{alias}(\omega)$ *é dado por:*

$$P_{alias}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(j(\omega + 2\pi k) \frac{1}{h} \right) \right|^2} \tag{1.4}$$

A prova deste teorema é apresentada em (KAO; LINCOLN, 2004).

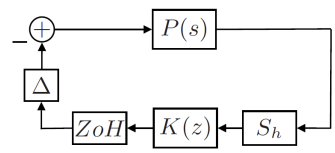


Figura 1.10: Malha de Controle Formada por: uma Planta Contínua $P(s)$, um Amostrador S_h, um Controlador Discreto $K(z)$, um Sustentador de Ordem Zero ZoH e um Atraso Variável Δ

Definição 1.3.2 (Margem de Jitter) *Dado o sistema da Figura 1.10, a margem de jitter (*$\hat{J}_m(L)$*) é definida como o maior valor para o qual a estabilidade de malha fechada é garantida para qualquer atraso* $\Delta \in [L, L + \hat{J}_m(L)]$.

A garantia da estabilidade nominal não é uma condição de projeto satisfatória, uma vez que os sistemas reais estão sujeitos a incertezas de modelagem. Assim, é necessário projetar sistemas com certa robustez, de modo que os mesmos não sejam afetados significativamente por perturbações ou erros de modelagem.

A margem de fase aparente $\hat{\varphi}_m$ é uma proposta que tenta contemplar o fenômeno do atraso variável, sendo definida da seguinte forma:

Definição 1.3.3 (Margem de Fase Aparente) *Dado o sistema da Figura 1.10 e assumindo que são conhecidos* L *e* J*. A margem de fase aparente é definida como o maior valor de* $\hat{\varphi}_m$ *para o qual a estabilidade é garantida para qualquer atraso* $\Delta \in [L + \hat{\varphi}_m/\omega_c, L + \hat{\varphi}_m/\omega_c + J]$.

Em NCS, usando os conceitos de Margem de *Jitter* e de Margem de *Fase Aparente*, para garantir a estabilidade do sistema é suficiente satisfazer à seguinte condição:

$$R_{loop}^{max} \leq J_m(R_{loop}^{min} + \frac{\hat{\varphi}_m}{\omega_c}) + R_{loop}^{min} \tag{1.5}$$

em que R_{loop}^{min} e R_{loop}^{max} representam os atrasos mínimo e máximo na atuação em um laço de controle.

O R_{loop}^{min} é inteiramente dependente da configuração. Em um sistema de controle via rede, no qual os dispositivos sensor, controlador e atuador se comunicam através da rede, o melhor caso para o atraso na atuação em um laço de controle (R_{loop}^{min}) pode ser obtido quando os tempos de computação no melhor caso são observados nos dispositivos e não há interferência de comunicação interna ou externa no sistema. As interferências de comunicação externas ocorrem quando mensagens de outros sistemas interferem na transmissão das mensagens do sistema analisado. As interferências de comunicação internas acontecem quando mensagem de rodadas anteriores do laço de controle interferem na transmissão de mensagens de rodadas subsequentes. O R_{loop}^{max} se verifica quando os tempos de computação no pior caso são observados nos dispositivos e existem interferências internas e externas de comunicação.

1.3.2 Qualidade de Controle (QoC)

Para análise de desempenho do NCS, é utilizada a métrica Qualidade de Controle (QoC). Uma abordagem clássica para verificar a QoC do sistema é defini-la em função do erro do sistema em malha fechada baseada na Integral Absoluta do Erro (IAE, *Integral of Absolute Error*), isto é:

$$QoC = \frac{1}{IEA} \tag{1.6}$$

em que IAE é definido como:

$$IAE = \int_0^\infty |e(t)|\, dt \tag{1.7}$$

Um bom desempenho está associado a um baixo valor do IAE e, consequentemente, um desempenho ruim está associado a um alto valor do IAE.

Uma outra abordagem mais recente para calcular a QoC de uma NCS, é comparar a robustez do NCS em relação a um sistema de controle tradicional. Para isto, compara-se a *margem de fase aparente* ($\hat{\varphi}_m$) do sistema sob atrasos variados com a margem de fase (φ_m) de um sistema tradicional. Nesta abordagem, quanto mais próximo o valor de $\hat{\varphi}_m$ for de φ_m, menor se espera que seja o efeito do atraso variável na degradação do sistema. Assim, pode-se definir uma relação de degradação (ou QoC), como:

$$QoC = \frac{\hat{\varphi}_m}{\varphi_m} \tag{1.8}$$

Mais adiante, na Seção 1.3.4, será discutido como a métrica de QoC, baseada em *margem de fase aparente*, pode ser usada para fornecer indicativos tanto de robustez quanto de desempenho em NCS construídas usando rede Ethernet Comutada.

1.3.3 Procedimento para Avaliação dos Atrasos em Sistema de Controle via Rede Ethernet Comutada

A avaliação dos atrasos em NCS depende das características da rede de comunicação usada em seu projeto. Assim, como exemplo, esta Seção descreve um procedimento, proposto por de Sá e outros (2008), para a avaliação de atrasos em Sistema de Controle via Rede Ethernet Comutada – do inglês *Switched Ethernet* (TANENBAUM, 2003). Esse tipo de rede tem características não determinísticas (DECOTIGNIE, 2005). No entanto, pode-se limitar o tráfego segundo algumas condições, de forma que seja possível determinar o máximo atraso induzido pela rede – para uma discussão mais detalhada, consultar, por exemplo, Lee e Lee (2002) e Georges, Divoux e Rondeau (2005).

Geralmente, a degradação do desempenho provocada por redes Ethernet Comutada é obtida através de um procedimento analítico, como apresentado por Georges e outros (2006), ou quantificado via experimentação, como apresentado por Lee e Lee (2002). A abordagem analítica, usando a margem de fase aparente combinada com o uso de alguns conceitos de teoria das filas, como apresentado por de Sá e outros (2008), por exemplo, permite estabelecer os limites para o uso da rede, de modo a guiar o projeto do sistema, encontrar o número de plantas que podem ser controladas e verificar a degradação no desempenho das mesmas.

Em redes Ethernet Comutadas, quando uma mensagem é transmitida, a mesma é empacotada em um quadro *Ethernet*, encaminhada para o *buffer* de entrada do *Switch* Ethernet (associado ao dispositivo transmissor) para, em seguida, ser encaminha para o buffer de saída do Switch Ethernet (associado ao dispositivo receptor) – a Figura 1.11 apresenta o modelo em filas representando a arquitetura interna de um *Switch* Ethernet.

Para o modelo de filas da Figura 1.11, sendo C_x a capacidade de transferência do *buffer* (em bits por segundo) e S_{msg} o tamanho (em bits) do maior quadro *Ethernet* transmitido pelos dispositivos do sistema de controle, a velocidade de atendimento (μ_x) em quadros por segundo (qps), será:

$$\mu_x = \frac{C_x}{S_{msg}}$$

Assumindo fluxos periódicos, a taxa total de quadros (λ_x) que chegam um *buffer* x do *Switch* será o somatório N_x de todos os quadros, que são encaminhadas por dispo-

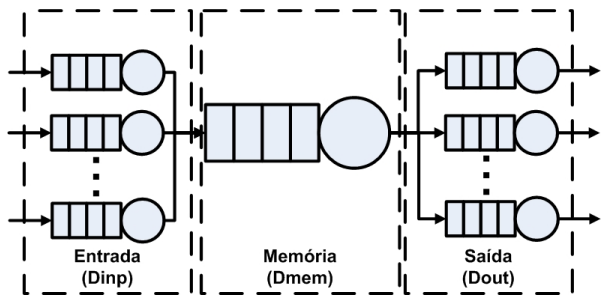


Figura 1.11: Modelo em Rede de Filas do Switch

sitivos (ou *buffers*) a tal *buffer* (em um cenário de pior caso) dentro de um intervalo de observação T, assim:

$$\lambda_x = \frac{N_x}{T}$$

É necessário que $\lambda_x < \mu_x$ para que a capacidade do *buffer* não seja comprometida, ocasionando estouro de *buffer* ou perda de mensagens.

Então, através da lei de *Little* (COOPER, 1981) é possível estimar o tempo de resposta (D_x) de um quadro através um *buffer* x:

$$D_x = \frac{N_x}{\mu_x}$$

Assim, assumindo o modelo apresentado na Figura 1.11, o atraso $D(i,j)$ observado na transmissão de uma mensagem do dispositivo i para um dispositivo j será:

$$D(i,j) = D_{inp}(i) + D_{mem} + D_{out}(j) \tag{1.9}$$

em que: $D_{inp}(i)$ representam o atraso do quadro no *buffer* de entrada i (associado ao transmissor); D_{mem} representa o atraso no processo de comutação (associada a memória compartilhada do *Switch*); e $D_{out}(j)$ representa o atraso no *buffer* de saída j (associado ao receptor da mensagem).

O fluxo máximo de mensagens ocorre ciclicamente dentro de um determinado intervalo de tempo, dito hiper-período ou executivo cíclico, que corresponde ao mínimo múltiplo comum (mmc) de todos os períodos dos nós transmissores que integram a rede. Uma análise de pior caso do atraso fim-a-fim deve ser feita assumindo o intervalo de observação T dentro do hiper-período.

1.3.4 Exemplo Motivador: Avaliação de Desempenho de Sistema de Controle via Rede Ethernet Comutada

Como exemplo motivador, nesta seção, os conceitos apresentados na seção anterior serão utilizados para avaliação de desempenho de NCS via rede Ethernet Comutada. Para isto, um modelo de NCS é discutido, em seguida, este modelo é avaliado a partir de abordagens experimental e analítica.

Modelo de Sistema

Considera-se um modelo de sistema em que os dispositivos sensor, controlador e atuador se comunicam através de uma rede Ethernet Comutada. Nesta rede, o *Switch* é não bloqueante, assim o atraso de comutação é desprezível. Por outro lado, os *buffers* de entrada e saída do *Switch* são simétricos e os atrasos em tais *buffers* são determinados pelas velocidades das portas.

Os dispositivos do sistema são dotados de capacidade de processamento e utilizam um sistema operacional de tempo real, o qual escalona as tarefas usando uma política baseada em prioridades fixas, seguindo o seguinte modelo: a tarefa τ_{sns} é executada no sensor, é ativada periodicamente a cada h unidades de tempo e a cada ativação envia o estado da planta para o dispositivo controlador; a tarefa τ_{ctrl}, por sua vez, é executada no controlador, é ativada após o recebimento da informação do estado da planta (enviada pelo nó sensor) e, então, envia a ação de controle ao dispositivo atuador; e, por fim, o atuador executa a tarefa τ_{atd} toda vez que recebe uma ação de controle.

Existem n plantas no sistema e as tarefas de controle executam em um único nó da rede. Todavia, para cada planta estão associados um nó sensor e um nó atuador.

Avaliação Experimental

Parâmetros, Fatores e Métricas. Nas simulações e análises foram utilizadas as mesmas plantas e os mesmos controladores de Santos, Moreno e Montez (2007b), os quais são apresentados na Tabela 1.1.

Tabela 1.1: Plantas e controladores utilizados no sistema

Planta	Controlador
$P_1(s) = \dfrac{-1064}{(s-36.16)(s+36.16)}$	$K_1(z) = \dfrac{-220.5z^2 + 417z - 197.1}{z^2 - 1.567z + 0.567}$
$P_2(s) = \dfrac{4 \cdot 10^4}{(s-200)(s+200)}$	$K_2(z) = \dfrac{27.16z^3 - 33.5z^2 + 9.869z - 0.725}{z^3 - 0.0845z^2 - 0.0704z + 0.0671}$
$P_3(s) = \dfrac{5 \cdot 10^7}{s(s^2 + 100s + 2.5 \cdot 10^5)}$	$K_3(z) = \dfrac{7.624z^4 - 15.93z^3 + 10.52z^2 - 1.85z + 0.0028}{z^4 + 0.7321z^3 + 0.2844z^2 - 0.1446z + 0.0174}$

Na Tabela 1.2, as propriedades temporais das tarefas do sistema são representadas pelo tempo de computação no pior caso ($WCET$) e pelo período de amostragem (h).

Tabela 1.2: Propriedades temporais das tarefas do sistema

Planta	Controlador	Tarefa	h	$WCET$
$P_1(s)$	$K_1(z)$	τ_{sns}	680	13
		τ_{ctrl}	-	1
		τ_{atd}	-	27
$P_2(s)$	$K_2(z)$	τ_{sns}	690	14
		τ_{ctrl}	-	1
		τ_{atd}	-	28
$P_3(s)$	$K_3(z)$	τ_{sns}	650	12
		τ_{ctrl}	-	1
		τ_{atd}	-	24

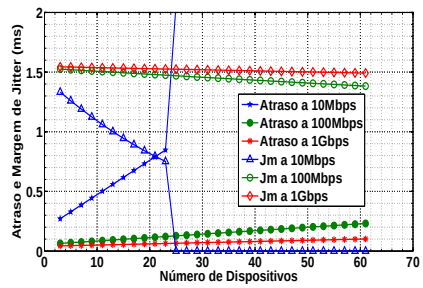
Para avaliação de desempenho, são assumidos os seguintes parâmetros e fatores: Velocidade da rede (10, 100 e 1000 $Mbps$); Modelo de comutação usada no *Switch* (*store-and-forward*); Tamanho do *buffer* de mensagens (10000 quadros); Estratégia em caso de estouro de *buffer* (descarte de quadros); Tamanho dos quadros (576 *bytes*); Número de plantas ($n = 1$ a $n = 60$).

Durante as avaliações, observa-se a qualidade do controle a partir da margem de fase aparente. Essa métrica é correlacionada com o número de dispositivos que competem pelo uso da rede. Além disso, são apresentados os comportamentos dos atrasos de comunicação e de computação em função do aumento do número de dispositivos na rede. Os dados apresentados foram todos obtidos em cenários de pior caso (hiperperíodo), o qual se verifica no mmc entre os períodos de amostragem (i.e., mmc entre 680, 690 e $650ms$).

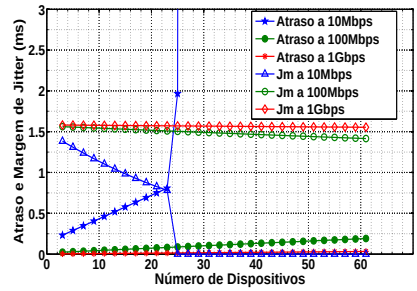
Resultados dos Experimentos. As Figuras 1.12(a), 1.12(b) e 1.12(c) apresentam, respectivamente, o comportamento dos atrasos na malha de controle, de comunicação e de computação em função do número de dispositivos interagindo na rede. O atraso na malha de controle equivale aos atrasos de comunicação e de computação somados.

Para taxas nominais de transferência de $10Mbps$ a medida que o número de dispositivos aumenta, o atraso na malha de controle aumenta exponencialmente, enquanto a *margem de jitter* decai na mesma proporção. Para taxas de 100 e $1000Mbps$, por outro lado, o atraso na malha de controle e as *margens de jitter* do sistema são menos sensíveis ao aumento do número de dispositivos que compartilham a rede.

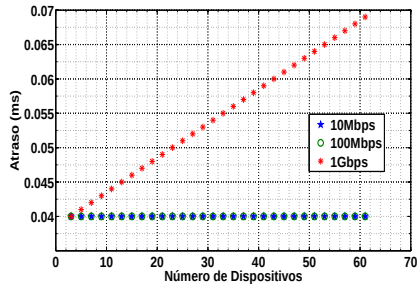
Ao correlacionar os dados apresentados nas Figuras 1.12(a), 1.12(b) e 1.12(c), é possível perceber que o atraso na comunicação é o componente principal do atraso na malha de controle (para a configuração de sistema proposta). Enquanto o atraso de



(a) Atrasos na malha de controle (computação + comunicação)



(b) Atraso de comunicação



(c) Atraso de computação

Figura 1.12: Atrasos e margem de *jitter* em função do número de dispositivos

comunicação varia em décimos de milissegundos, o atraso de computação varia em frações de centésimos de ms.

Um efeito interessante pode ser observado quando a taxa nominal de comunicação é elevada ($1Gbps$). Para taxas de comunicação de $10Mbps$ e $100Mbps$ a disputa pelo acesso ao meio faz com que as mensagens de sensoriamento cheguem ao controlador espaçadas entre $5.76\mu s$ (para taxas de $100Mbps$) e $57.6\mu s$ (para taxas de $10Mbps$). Isso provoca um maior espaçamento entre as ativações das tarefas de controle no controlador. Para $1Gbps$ por outro lado, o tempo de propagação das mensagens na rede é de apenas $0.576\mu s$, o que diminui o espaçamento entre as ativações das tarefas no controlador e faz com que haja uma disputa maior das tarefas pelo uso da capacidade de processamento de tal dispositivo.

A Qualidade do Controle (QoC, *Quality of Control*) em função do aumento do tráfego na rede de comunicação é apresentada na Figura 1.13. Analisando esta Figura em conjunto com a Figura 1.12(a), é possível perceber que à medida que o tráfego na rede aumenta a qualidade do controle diminui, tal efeito é menos severo à medida em que a taxa nominal de transferência da rede aumenta. Na maioria dos experimentos, as

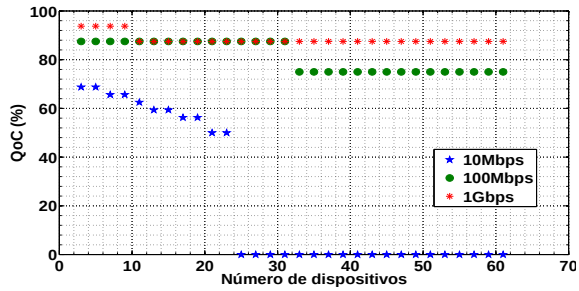


Figura 1.13: Impacto na QoC

plantas se mantiveram estáveis, exceto nos experimentos com taxas nominais de transferência da rede de $10Mbps$ e com número de dispositivos superior a 23 (ver Figura 1.13). É possível notar que quando o controle é instável, a qualidade do desempenho do sistema é zero. É interessante observar que a QoC calculada em função da margem de fase aparente varia de forma discreta em função do aumento do número de dispositivos.

Abordagem Analítica

Através da estratégia proposta na Seção 1.3.3, é possível predizer os resultados obtidos nas simulações. Para a análise, considerou-se o cenário em que as portas do switch possui $10Mbps$ (i.e., $\mu_{inp} = \frac{10 \cdot 10^6}{576} fps$) e com uma configuração em que a rede é compartilhada por 11 plantas. No instante crítico, cada sensor envia uma mensagem ao controlador, assim o atraso associado a cada entrada do *Switch* no qual cada sensor está ligado será:

$$D_{inp}(sns) = \frac{576}{10 \cdot 10^6} = 0.0576ms$$

Na saída do *Switch* para o controlador, por sua vez, tem-se:

$$D_{out}(ctrl) = \frac{576}{10 \cdot 10^6} * 11 = 0.6336ms$$

Das mensagens que chegam ao controlador, cada uma é encaminhada ao atuador de sua respectiva planta, assim:

$$D_{inp}(ctrl) = D_{out}(atd) = 0.0576ms$$

O atraso na comunicação é dado por:

$$D(sns, ctrl) + D(ctrl, atd) = 3 * 0.0576 + 0.6336 = 0.8064ms$$

Observe que em um cenário como esse, no melhor caso, os quadros dos diversos dispositivos chegam no controlador espaçados de $0.0576ms$. Portanto, verifica-se que não existe interferência entre as tarefas de controle, uma vez que o tempo de computação no pior caso das tarefas de controle são iguais a $1\mu s$.

Com isso, o atraso máximo na malha de controle para esse caso será:

$$R_{loop}^{max} = R_{sns} + R_{ctrl} + R_{atd} + 0.8064ms$$

Para a malha $P_1(s)K_1(z)$, por exemplo, o atraso máximo será:

$$R_{loop}^{max} = 0.8064ms + 0.041ms = 0.8474ms$$

Através dessa abordagem combinada com a margem de fase aparente é possível estabelecer os limites para o número de plantas no sistema, por exemplo: 128 e 1302 plantas para as taxas nominais de $100Mbps$ e $1Gbps$, respectivamente, considerando as configurações apresentadas.

1.4 Análise do Desempenho de NCS sujeitos a Perdas de Mensagens

Nesta seção é avaliado o desempenho de um NCS utilizando um filtro de Kalman como observador para estimar os estados quando a medida de saída que é enviada para o controlador, por meio de uma rede de comunicação, é perdida ou está atrasada.

A abordagem utilizada para tratar perda e atrasos de mensagens é a mesma proposta por Sinopoli e outros (2004). Os atrasos serão simulados para redes cabeadas, com protocolos CAN e Ethernet, enquanto as perdas de mensagens serão simuladas em redes sem fio com protocolo IEEE 802.11b.

1.4.1 Filtragem de Kalman para sistemas discretos

O Filtro de Kalman (1960) é um método iterativo, capaz de estimar as variáveis de estado de sistemas dinâmicos representados por equações lineares. O método é dito iterativo quando é possível estimar os parâmetros de um determinado modelo à medida que os dados do processo são disponibilizados. A principal vantagem em usar esse método é sua eficiência computacional em comparação com métodos clássicos. Além disso, esse filtro é o fato de atualizar os cálculos a cada nova medida fornecida pelo sistema observado, não sendo necessário conhecer a priori todas as medidas anteriores.

O filtro de Kalman pressupõe que o sistema seja perturbado por ruídos brancos e gaussianos, de forma que os estados possam ser tratados como variáveis aleatórias gaussianas. A formulação do filtro consiste em incorporar toda informação obtida a partir de medições ruidosas para estimar as variáveis desejadas. A estimativa encontrada pelo filtro, a partir de medidas afetadas por ruídos, é ótima, pois minimiza os quadrados dos erros. A estimação de estados de um sistema discreto linear invariante no tempo é definida pelas seguintes equações:

$$\begin{aligned} x_{k+1} &= A * x_k + B * u_k + w_k \\ y_k &= C * x_k + v_k \end{aligned} \tag{1.10}$$

em que, $x_k \in \Re^n$ é o vetor de estados, $y_k \in \Re^m$ é a saída do sistema e $u_k \in \Re^p$ representa o vetor de entradas. As variáveis $w_k \in \Re^n$ e $v_k \in \Re^m$ são, respectivamente, ruídos Gaussianos do processo e da medida – ambos independentes, de média nula e com matrizes de covariância $Q \geq 0$ e $R > 0$, respectivamente. Sendo essas variáveis caracterizadas pelas seguintes distribuições probabilísticas $p(w) \approx N(0, Q)$ e $p(v) \approx N(0, R)$.

1.4.2 Caracterização Computacional do Filtro de Kalman

No emprego do filtro de Kalman, estuda-se o desenvolvimento matemático da estimação de estados predita ou estimação *a priori* ($\hat{x}_{k+1|k}$), do vetor de estados no instante $k+1$ obtida a partir de informação disponível no instante k.

Esta estimação pode ser calculada recursivamente através de uma equação que faz a correção do erro estimado dando um ganho de desempenho do filtro. A seguir, são desenvolvidas as equações da estimação de estado filtrada (ou estimação *a posteriori*), com base na leitura feita no instante $k+1$. Esta estimação é calculada através de uma equação (Equação 1.15) que faz a correção do erro estimado dando um ganho de desempenho do filtro. É possível definir a partir disto o erro de estimação a *priori* (Equação 1.11) e a *posteriori* (Equação 1.12):

$$e_{k+1|k} = x_{k+1} - \hat{x}_{k+1|k} \tag{1.11}$$

$$e_{k+1|k+1} = x_{k+1} - \hat{x}_{k+1|k+1} \tag{1.12}$$

As matrizes de covariância associadas a $e_{k+1|k}$ e a $e_{k+1|k+1}$ são, respectivamente:

$$P_{k+1|k} = E[e_{k+1|k} * e'_{k+1|k}] \tag{1.13}$$

$$P_{k+1|k+1} = E[e_{k+1|k+1} * e^{'}_{k+1|k+1}] \tag{1.14}$$

em que a operação $M^{'}$ denota a transposta de uma matriz M e $E[.]$ denota o valor esperado condicional dadas as informações até o instante k.

Na técnica usando o filtro de Kalman, deve-se estimar o estado a *posteriori*, a partir da combinação linear de uma estimativa a *priori* com uma diferença ponderada entre a medida atual e uma predição da medida, como mostra a Equação 1.15.

$$x_{k+|k+1} = \hat{x}_{k+1|k} + K_k * (y_k - C * \hat{x}_{k+1|k}) \tag{1.15}$$

A matriz K_k é denominada ganho de Kalman e é obtida por meio da minimização da matriz de covariância a *posteriori*, sendo sua expressão dada por:

$$K_k = P_{k+1|k} C^{'} (C P_{k+1|k} C^{'} + R)^{-1} \tag{1.16}$$

1.4.3 Algoritmo para o Filtro de Kalman Discreto

O algoritmo para o filtro de Kalman compreende duas etapas fundamentais predição e correção, para as quais temos um conjunto de equações a serem aplicadas: equações de atualização de tempo e equações de atualização de medida. As equações de atualização de tempo são responsáveis por projetar adiante o estado atual, usando um estimador da covariância do erro para obter uma estimativa a *priori* para o próximo instante. As equações de atualização de medida são responsáveis pela realimentação, isto é, incorporar uma medida nova na estimativa a *priori* para obter um estimador melhorado a *posteriori*. As equações 1.17 e 1.18 são específicas para as atualizações de tempo, enquanto que as equações 1.20 e 1.21 são específicas as atualizações da medida.

$$\hat{x}_{k+1|k} = A * \hat{x}_{k|k} + B * u_k \tag{1.17}$$

$$P_{k+1|k} = A * P_{k|k} * A^{'} + Q \tag{1.18}$$

$$K_k = P_{k+1|k} * C^{'} * (C * P_{k+1|k} * C^{'} + R)^{-1} \tag{1.19}$$

$$x_{k+1|k+1} = \hat{x}_{k+1|k} + K_k * (y_k - C * \hat{x}_{k+1|k}) \tag{1.20}$$

$$P_{k+1|k+1} = (I - K_k * C) P_{k+1|k} \tag{1.21}$$

A Figura 1.14 apresenta um fluxograma com o algoritmo do Filtro de Kalman, para as condições iniciais $\hat{x}_{0|0}$ e $P_{0|0}$.

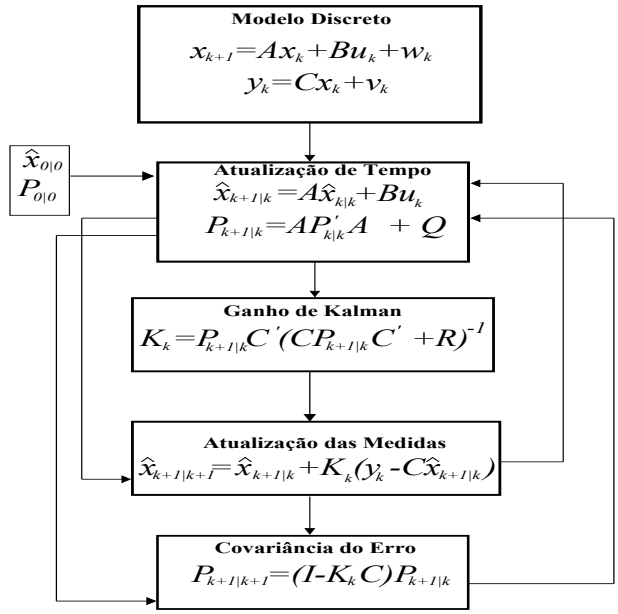


Figura 1.14: Implementação do algoritmo do Filtro de Kalman

1.4.4 Filtragem de Kalman para uma rede não confiável

Nesta seção é considerado o problema de estimação de estados para um Sistema de controle via rede de comunicação não confíavel – isto é, sujeito a perdas de mensagens e atrasos variados. A análise realizada é baseada na proposta de Sinopoli e outros (2004). Do ponto de vista da teoria de controle, Sinopoli e outros (2004) argumentam que atrasos significativos são equivalentes a perdas, já que os dados precisam chegar a tempo de serem usados pelo controlador e assim não afetar o desempenho do sistema.

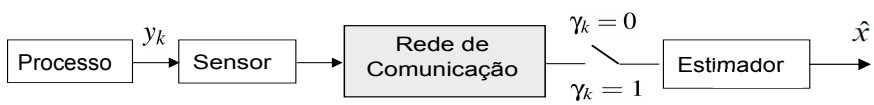


Figura 1.15: Esquema geral da transmissão de dados em uma rede de comunicação em que podem ocorrer perdas de mensagens

No cenário proposto por Sinopoli e outros (2004), ver Figura 1.15, o sensor fornece medidas ao estimador por meio de uma rede não confiável, que estima os dados e os fornece ao controlador. A perda de mensagem é representada através de uma variável aleatória γ_k que indica se a medida foi ou não entregue corretamente, isto é: se $\gamma_k = 0$, então a mensagem foi perdida; caso contrário, $\gamma_k = 1$.

Sinopoli e outros (2004) mostraram que, mesmo com perdas de mensagens, o Filtro de Kalman ainda é ótimo. Entretanto, as equações originais do filtro precisam ser modificadas para considerar perda de mensagens, resultando nas equações abaixo:

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k} \tag{1.22}$$

$$P_{k+1|k} = AP_{k|k}A^{'} + Q \tag{1.23}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \gamma_{k+1}K_{k+1}(y_{k+1} - C\hat{x}_{k+1|k}) \tag{1.24}$$

$$P_{k+1|k+1} = P_{k+1|k} - \gamma_{k+1}K_{k+1}CP_{k+1|k} \tag{1.25}$$

$$K_{k+1} = P_{k+1|k}C^{'}(CP_{k+1|k}C^{'} + R)^{-1} \tag{1.26}$$

As equações (1.22) e (1.23) são as atualizações de tempo (predição), as equações (1.24) e (1.25) são as atualizações das medidas (correção) e $P_{k+1|k}$ e $P_{k+1|k+1}$ são as covariâncias *a priori* e *a posteriori*, respectivamente.

Sinopoli e outros (2004) analisam também as propriedades de convergência estatísticas da covariância do erro, mostrando a existência de um valor crítico para a chegada de mensagens, isto é, para o qual o erro diverge se ultrapassar esse valor. Para isto, realizam uma análise teórica do problema sem se preocupar, por exemplo, com o tipo de protocolo usado pela rede de comunicação.

1.4.5 Exemplo Motivador: Avaliação de Desempenho de Sistema de Controle via Rede Não Confiável

Como exemplo motivador, nesta seção, são realizadas simulações para analisar o desempenho de um NCS baseado em presença de um Filtro de Kalman e sujeito a atrasos e perdas de mensagens. A avaliação é realizada considerando três tipos de rede: (I) CAN, com taxa de transmissão de 1Mbps; (II) Ethernet com taxa de transmissão de 1000Mbps; e (III) IEEE 802.11b com taxa de transmissão de 52Mbps.

As simulações são realizadas com o auxilío do TrueTime (HENRIKSSON; CERVIN; ÅRZÉN, 2002). No ambiente simulado, as medições são realizadas periodicamente por um sensor e então transmitidas através da rede de comunicação ao controlador. Os modelos da planta e do controlador, Equações 1.27 e 1.28, respectivamente, são os mesmos propostos por Perez, Moreno e Montez (2006).

$$P_1(s) = \frac{-1064}{(s - 36.16)(s + 36.16)} \tag{1.27}$$

$$K_1(z) = \frac{-214z^2 + 399.9z - 186.7}{z^2 - 1.48z + 0.4797} \tag{1.28}$$

O controlador proposto por Perez, Moreno e Montez (2006) foi obtido a partir do procedimento de *co-design* com alocação de pólos para o período de amostragem de $h_1 = 0,85ms$. Os parâmetros do Filtro de Kalman são dados a seguir e foram determinados por meio de simulações.

$$Q = \begin{bmatrix} 500 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 0 \end{bmatrix}, \hat{x}_{0|0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, P_{0|0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{1.29}$$

Dois casos de estudo comparando a resposta do NCS com e sem filtro de Kalman são apresentados. No primeiro caso, é analisado o desempenho do NCS para redes cabeadas com atrasos aleatórios na entrega de mensagens. No segundo caso, são consideradas perda de mensagens na comunicação entre dispositivo usando rede sem fio.

Implementação do NCS baseado no Filtro de Kalman. A cada período de amostragem h_1, o sensor coleta uma amostra da planta e envia pela rede para o controlador. Se o controlador não recebe a amostra enviada antes do próximo período de amostragem, a mesma é considerada perdida e o controlador ativa o estimador baseado no Filtro de Kalman. Então, o estimador calcula e repassa o valor estimado da amostra para o controlador. A cada amostra recebida, o controlador também ativa o estimador para que este possa atualizar e armazenar suas variáveis de relacionadas a estimativa de tempo e de medida. A execução do estimador é realizada da seguinte forma:

1. Inicializa o estimador com as estimativas iniciais $\hat{x}_{0|0}$ e $P_{0|0}$,
2. Atualiza variáveis de tempo:
   - $\hat{x}_{k+1|k} = A\hat{x}_{k|k}$
   - $P_{k+1|k} = AP_{k|k}A^{'} + Q$
3. Atualiza variáveis de medida:
   - Calcula o Ganho de Kalman: $K_{k+1} = P_{k+1|k}C^{'}(CP_{k+1|k}C^{'} + R)^{-1}$
   - Atualiza a variável de estado: $\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \gamma_{k+1}K_{k+1}(y_{k+1} - C\hat{x}_{k+1|k})$
   - Atualiza a matriz de covariância: $P_{k+1|k+1} = P_{k+1|k} - \gamma_{k+1}K_{k+1}CP_{k+1|k}$
4. Calcula a saída estimada $y_e = C\hat{x}_{k+1|k+1}$ e informa ao controlador.

Avaliação sobre redes cabeadas: CAN e Ethernet

Neste cenário, serão consideradas as redes cabeadas com protocolos de acesso ao meio CAN e Ethernet. O controlador, o sensor, o atuador e nós de interferência estão todos em dispositivos distintos compartilhando uma mesma rede. O estimador receberá as medidas enviadas pelo sensor através da rede, como é mostrado na Figura 1.16.

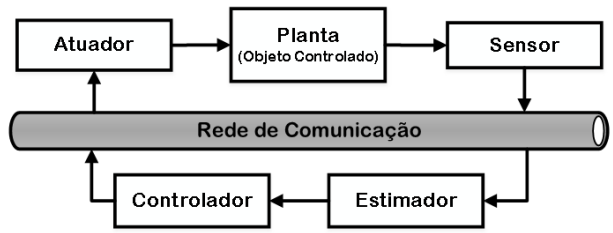


Figura 1.16: Topologia da rede para a implementação do projeto NCS com estimador

A carga adicional inserida na rede pelos nós de interferência afeta o desempenho do sistema de controle via rede. Entrentato, o uso do estimador melhora o desempenho do NCS – isto porque uma medida, ainda que estimada, estará disponível no controlador para executar o algoritmo de controle em um determinado período de amostragem.

Na situação proposta, h_1 é período de amostragem da planta e h_2, h_3, h_4, h_5 e h_6 são os períodos de amostragem das cinco tarefas de interferência periódicas que foram adicionadas à rede. Ainda existem mais duas tarefas de interferência que são dirigidas a eventos, pois recebem mensagens das tarefas periódicas h_2 e h_3 e enviam novamente para elas as medidas através da rede. A Tabela 1.3 apresenta os valores dos períodos de amostragem utilizados.

Tabela 1.3: Intervalos de ativação das mensagens em milissegundos

Cenário	h_1	h_2	h_3	h_4	h_5	h_6
1	0,85	0,84	3,7	1,1	1,1	1,1
2	0,85	0,84	3,7	0,6	0,48	0,54

Foram criados dois cenários de simulação a fim de gerar atrasos na rede mantendo a planta estável, isto é, de modo que os atrasos não gerem perda de *deadlines*, somente degradem o desempenho da planta. No segundo cenário, os períodos de amostragem foram diminuídos, para que deste modo atrasos maiores sejam gerados.

Na Figura 1.17 são representados os escalonamentos das mensagens enviadas do sensor para o estimador com os protocolos CAN e Ethernet. A representação se dá por meio de três níveis: o nível baixo representa que não se deseja enviar uma mensagem,

o nível médio significa que o nó aguarda a liberação do barramento para tentar enviar a mensagem e o nível alto representa o momento que a mensagem está sendo transmitida. Por meio da Figura 1.17, nota-se que para o protocolo Ethernet ocorrem poucos níveis médios e as mensagens são transmitidas com certa regularidade. No entanto, para redes CAN, essas transmissões não se apresentam periodicamente e ocorrem níveis médios com maior frequência mostrado mais espera para transmitir as mensagens, portanto os atrasos são mais frequentes durante as transmissões. Com atrasos maiores, devido a menor taxa de transmissão, o desempenho do sistema com rede CAN apresentará uma degradação mais significativa.

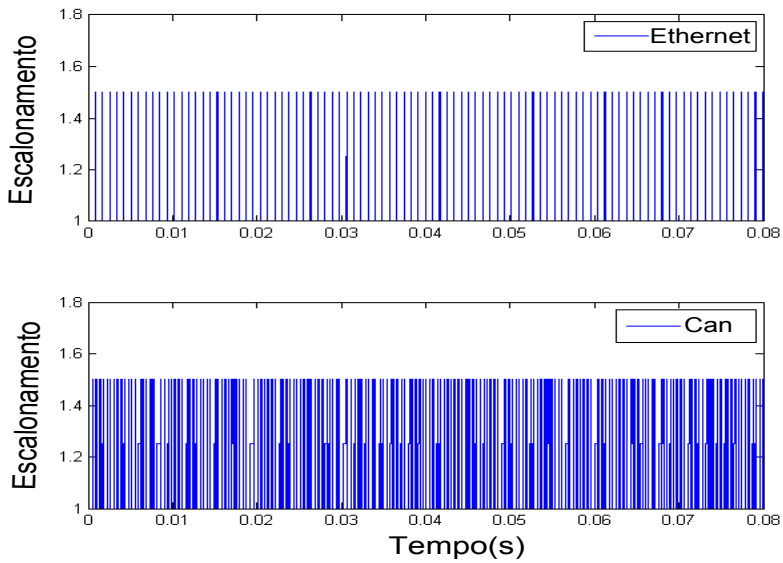


Figura 1.17: Escalonamento das medidas entregues no estimador

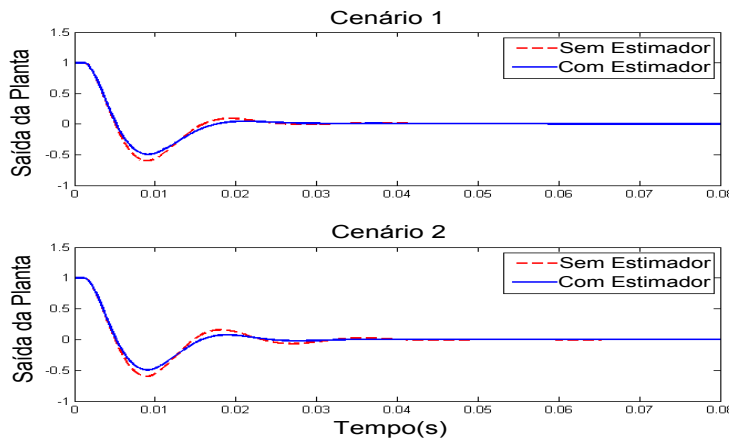
A fim analisar o desempenho do sistema, a Qualidade do Controle (QoC) é calculada para o NCS com e sem o estimador. A Tabela 1.4 apresenta os resultados das simulações nos dois cenários propostos para as redes CAN e Ethernet.

Tabela 1.4: QoC da planta com e sem estimador

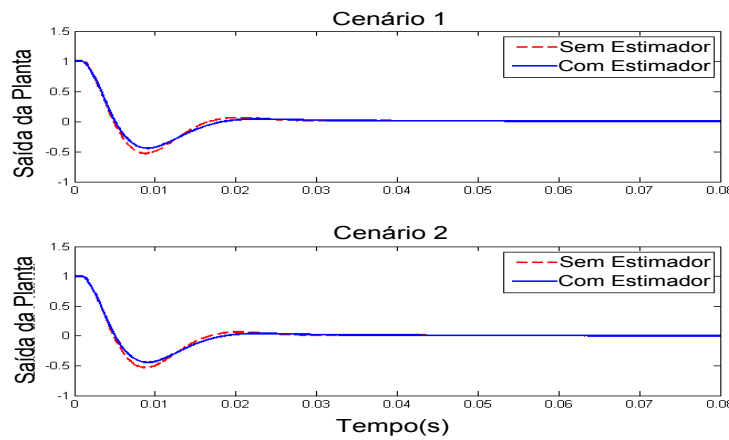
Protocolo	Cenário	Com Estimador	Sem Estimador
CAN	Cenário 1	129,9990	123,3706
	Cenário 2	133,6557	118,3882
Ethernet	Cenário 1	135,9120	131,1034
	Cenário 2	135,9893	131,2639

Analisando a Tabela 1.4, nota-se que a QoC em rede Ethernet é maior e, portanto, tem melhor desempenho. Além disso, a presença do estimador em um NCS com atrasos melhora o desempenho do sistema, já que a QoC aumenta quando o estimador é inserido no sistema para os dois protocolos na presença de atrasos.

Pode-se observar ainda que a partir das respostas em malha fechada apresentadas nas Figuras 1.18(a) e 1.18(b) a degradação sofrida pelas malhas é amenizada com a presença do estimador, compensando assim a presença dos atrasos aleatórios.



(a) Saída da planta para NCS/CAN



(b) Saída da planta para NCS/Ethernet

Figura 1.18: Saída da planta para redes CAN e Ethernet (com e sem estimador)

Avaliação sobre rede não confiável: IEEE 802.11b com taxas de 20% e 40% de perda de mensagens

Neste cenário, simula-se uma rede sem fio, com protocolo IEEE 802.11b. Analisaremos o desempenho de controle para dois casos de perda de mensagens, com taxas de perda 20% e 45%. O sistema planta/controlador utilizado será o mesmo do cenário anterior. No entanto, o cenário será composto apenas pelo NCS e não serão consideradas tarefas de interferências já que o interesse é avaliar perdas de mensagens. Para se obter as taxas de perdas especificadas no projeto, utilizou-se uma distribuição de Bernoulli de espaço amostral $\{0, 1\}$, onde 0 indica a perda e 1 a entrega com sucesso. A topologia de rede é apresentada na Figura 1.19.

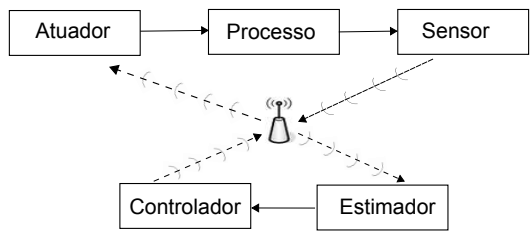


Figura 1.19: Topologia de uma rede sem fio

Conforme os resultados da QoC apresentados na Tabela 1.5, é possível perceber a melhoria no desempenho do NCS na presença do estimador.

Tabela 1.5: QoC do sistema com e sem estimador para uma rede sem fio com protocolo IEEE 802.11b

IEEE 802.11	Com Estimador	Sem Estimador
20% de perdas	133,9200	112,8296
45% de perdas	133,0083	76,9762

A Figura 1.20 mostra a resposta do sistema para os casos propostos.

Ao fazermos uma análise comparativa entre redes cabeadas e redes sem fio, é possível perceber que o desempenho do sistema fica mais comprometido na presença de perdas de mensagens. Além disso, é possível compensar a falta de informações (perda de mensagens) com o uso de um estimador (e.g., Filtro de Kalman), o qual se apresenta como uma ferramenta capaz de amenizar efeitos degenerativos do NCS e melhorar a QoC do sistema. Dentro deste contexto, ressaltamos a importância do conhecimento do tipo de protocolo utilizado para a troca de informações em uma rede de comunicação. Isto porque, na realização do projeto de um NCS, aspectos como perdas de mensagens

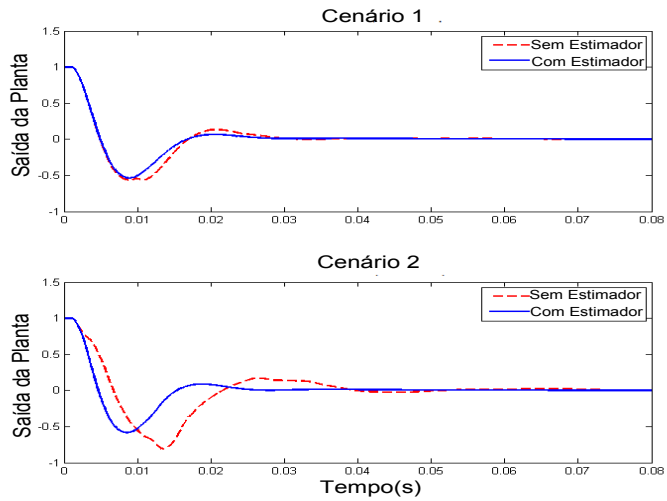


Figura 1.20: Saída da planta com e sem estimador para rede IEEE 802.11b

e atrasos de transmissão devem ser considerados e compensados de forma correta para que garantias de tempo real possam ser fornecidas ao sistema.

CAPÍTULO 2

Fundamentos de Detecção de Defeitos em Sistemas Distribuídos

Raimundo José de Araújo Macêdo e Alirio Santos de Sá

A capacidade de tolerar falhas é um requisito fundamental para sistemas de automação nos quais defeitos ou mal funcionamento do sistema podem implicar em prejuízos financeiros, ambientais, ou até mesmo perda de vidas humanas. Por melhores que sejam as técnicas usadas na construção destes sistemas, diversos fatores podem levar os mesmos a apresentarem defeitos: desgastes físicos de componentes, falhas humanas, fenômenos e/ou desastres naturais etc. Portanto, mecanismos adequados de tolerância a falhas devem ser empregados em tais sistemas. Os termos falha, erro, defeito, adotados neste livro, seguem o padrão de terminologia amplamente adotado pela comunidade científica internacional da área de tolerância a falhas (AVIZIENIS et al., 2004); sendo que *fault*, *error* e *failure* são traduzidos, respectivamente, para falha, erro e defeito.

Detecção de defeitos de componentes é um serviço utilizado na construção de mecanismos de tolerância a falhas. Este serviço consiste na verificação do funcionamento dos dispositivos ou componentes, de modo que, quando estes não se comportam de acordo com objetivos especificados, ações possam ser tomadas para substituição de componentes defeituosos, recuperação ou reconfiguração do sistema. A implementação dos detectores de defeitos depende do comportamento dos componentes em caso

de falhas – o qual é denominado de semântica de falhas (CRISTIAN, 1991). A semântica de falhas determina a técnica de verificação (ou monitoramento) a ser utilizada pelo serviço de detecção de defeitos. Por exemplo, se um dispositivo do sistema pode falhar produzindo valores incorretos, testes podem ser usados para comparar as saídas do dispositivo com saídas de referência. De outro lado, se um dispositivo falha somente omitindo valores de saída (temporária ou permanentemente), temporizadores podem ser usados para detectar defeitos (JALOTE, 1994).

A natureza descentralizada de sistemas de automação modernos tem impacto no projeto de detctores de defeitos. Em alguns desses sistemas, os dispositivos de automação estão distribuídos e interconectados a partir de redes de comunicação, por exemplo: Sistemas de Controle Sobre Rede (NCS, *Networked Control Systems*) – ver Capítulo 1; Sistemas de Controle Distribuídos (DCS, *Distributed Control Systems*) – ver Lian, Moyne e Tilbury (2002); aplicações distribuídas sobre Redes Veiculares Ad Hoc (VANET, *Vehicular Ad hoc Networks*) – ver Wang e Li (2009); Sistemas Ciber-Físicos (CPS, *Cyber-Physical Systems*) – ver Sha e outros (2009), entre outros. A natureza descentralizada traz uma série de benefícios, tais como a possibilidade da supervisão remota, a interconexão, a adição e remoção de novos dispositivos, a facilidade de integração entre sistemas etc. Todavia, nesses ambientes de automação distribuídos, as propriedades e o desempenho do serviço de detecção de defeitos dependem das garantias de qualidade de serviço (QoS, *Quality of Service*) entregues pelos sistemas de processamento e de comunicação utilizados. Por exemplo, se a infraestrutura de computação subjacente possui períodos em que não há garantias com relação aos tempos de processamento e transmissão das mensagens, o serviço de detecção de defeitos pode ser não confiável – fornecendo informações incorretas ou apresentando altos tempos de respostas na detecção de defeitos de componentes do sistema, o que pode comprometer o funcionamento de certos sistemas (ou subsistemas) de automação com requisitos temporais. Portanto, as hipóteses do modelo de sistema distribuído - isto é, o comportamento temporal e de falhas dos componentes de software, dispositivos e canais de comunicação - são usadas para determinar as propriedades do serviço de detecção de defeitos durante sua operação. Estas propriedades, ou garantias, influenciam, por sua vez, as características dos protocolos usados na construção de sistemas distribuídos robustos – como, por exemplo, consenso distribuído (GUERRAOUI et al., 2000) e difusão atômica (DÉFAGO; SCHIPER; URBÁN, 2004); duas técnicas usadas na construção de estratégias de replicação para tolerência a falhas (CRISTIAN, 1991).

O desempenho de um detector de defeitos é definido em termos de sua acurácia e rapidez. Para que este desempenho seja adequado, os parâmetros operacionais do detector de defeitos devem ser configurados de forma a atender as demandas das apli-

cações, mediante as condições de carga e disponibilidade dos recursos computacionais do ambiente distribuído. Uma vez que o detector de defeitos consome recursos computacionais para o processamento e transmissão das mensagens de monitoramento, se os intervalos entre verificações dos defeitos não são definidos de forma adequada, é possível que o detector degrade o desempenho da aplicação ou do próprio serviço de detecção, fornecendo informações incorretas ou tempos de detecção inadequados.

As demandas da aplicação em relação à qualidade do detector são expressas a partir de requisitos de QoS de detecção (CHEN; TOUEG; AGUILERA, 2002). Se as condições de carga e os requisitos de QoS são conhecidos a priori, é possível determinar, em tempo de projeto, parâmetros operacionais adequados para o detector de defeitos – é o que acontece em muitos NCS ou DCS nos quais os dispositivos possuem sistemas operacionais de tempo real e são interconectados por redes industriais com tempos de transmissão determinísticos. Se as condições de carga e os recursos disponíveis não são conhecidos a priori, mas possuem características que podem ser mapeadas a partir de distribuições de probabilidade definidas em tempo de projeto, é possível estabelecer estratégias dinâmicas de configuração dos parâmetros operacionais do detector para atender aos requisitos da aplicação (SÁ; MACÊDO, 2006, 2007). De outro lado, se as demandas das aplicações, as características de carga e/ou os recursos disponíveis variam durante a execução de forma não previsível, estratégias auto-configuráveis (ou autonômicas) precisam ser consideradas (SÁ; MACÊDO, 2010a, 2010b) – estas estratégias podem ser usadas em CPS nos quais redes de comunicação podem ser dinamicamente integradas à infraestrutura do sistema de automação, formando arranjos dinâmicos mais complexos e heterogêneos.

O restante deste capítulo discute diferentes aspectos relacionados à implementação de detectores de defeitos em sistemas de automação distribuídos. Assim, na Seção 2.1 serão apresentadas definições básicas sobre confiabilidade e disponibilidade em sistemas distribuídos. Mais adiante, nas Seções 2.2 e 2.3, são discutidos aspectos gerais sobre a implementação de detectores de defeitos em sistemas distribuídos, ressaltando questões relacionadas à QoS e à detecção adaptativa de defeitos. Todos estes conceitos são fundamentais para o entendimento dos aspectos técnicos discutidos no Capítulo 3 – o qual aborda a detecção de defeitos em ambientes distribuídos de automação.

2.1 Conceitos Básicos

Nesta Seção são apresentados conceitos básicos sobre confiabilidade e disponibilidade em sistemas distribuídos, para isto: a Seção 2.1.1 traz algumas definições e elementos

básicos relacionados à confiabilidade e disponibilidade de sistemas; A Seção 2.1.2 apresenta uma discussão sobre tipificação das falhas em componentes; a Seção aborda os níveis de confiabilidade de sistemas; e, por fim, a Seção 2.1.4 traz uma discussão sobre modelos de sistemas distribuídos.

2.1.1 Aspectos de confiabilidade e disponibilidade em sistemas distribuídos

Confiabilidade (do Inglês, *reliability*) e disponibilidade (do Inglês, *availability*) são propriedades que qualificam a confiança que se pode depositar no funcionamento correto de um sistema - também chamado de dependabilidade (do Inglês, *dependability*) do sistema (AVIZIENIS et al., 2004). Confiabilidade esta relacionada à capacidade do sistema se manter operacional, provendo continuamente um serviço correto, enquanto que disponibilidade se refere à capacidade do sistema estar disponível dado que o mesmo pode alternar entre operacional e não operacional. Como não há como se controlar todos os fatores que levam às falhas, confiabilidade e disponibilidae são melhores expressos em termos probabilisticos (CRISTIAN, 1991).

Falhas, *Erros* e *Defeitos* são fatores que devem ser considerados na implementação de sistemas confiáveis e disponíveis. Para caracterizar a ocorrência de um destes elementos, necessita-se de uma especificação bem definida dos requisitos do sistema.

Conceitualmente, um defeito denota um desvio entre o serviço que é entregue e o que foi especificado (AVIZIENIS et al., 2004). Tal desvio é originado de um estado errôneo (i.e., erro) ao qual é levado o sistema na presença de uma falha. Um erro pode permanecer latente até que algum evento no sistema promova a sua ativação. Como a falha é um evento indesejado que pode inserir um erro no sistema, pode-se afirmar que a falha é a causa indireta e primária do defeito – i.e., o evento de falha causa o estado de erro que, quando ativado, leva a um serviço defeituoso. Resumidamente, *Falha*, *Erro* e *Defeito* são abreviações para denotar, respectivamente, *Evento de falha*, *Estado de erro* e *Serviço defeituoso*.

Os conceitos de Defeito e Falha são relativizados pela organização hierárquica dos sistemas. De modo geral, um sistema é composto por outros sistemas (ditos subsistemas ou componentes). Da mesma forma, estes subsistemas são estruturados a partir de outros componentes e assim por diante, até se alcançar os subsistemas mais elementares (ou básicos), para os quais esta subdivisão não é evidente ou não tem um sentido próprio – ver Figura 2.1.

Neste contexto, o desvio no serviço provido por um subsistema (i.e., defeito) se torna uma falha para o sistema ao qual tal subsistema (ou componente) está inserido. Se

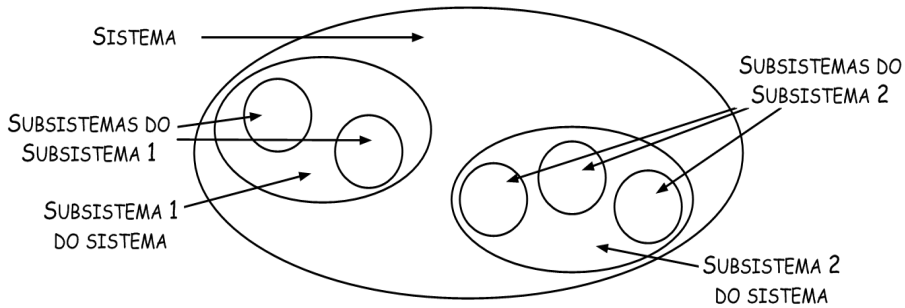


Figura 2.1: Hierarquia de sistema (JALOTE, 1994)

tal defeito no serviço do componente (ou falha no sistema) não for devidamente tratado, o sistema pode eventualmente deixar de entregar, para o sistema usuário, o serviço na forma em que foi especificado. Em outras palavras, as ocorrências de falha-erro-defeito se verificam em toda a hierarquia de um sistema, podendo uma falha interna em um componente básico se propagar para os subsistemas hierarquicamente superiores até atingir o serviço final - onde o defeito de um subsistema é caracterizado como falha no subsistema hierarquicamente imediatamente superior e assim por diante. A Figura 2.2, adaptada de Deswarte, Kanoun e Laprie (1998), exemplifica esse encadeamento entre falhas, erros e defeitos.

Figura 2.2: Cadeia de falha, erro e defeito

2.1.2 Classificação das falhas

A tipificação das falhas é um aspecto primordial no projeto de um sistema. As falhas podem ser tipificadas de diferentes formas, ver Avizienis e outros (2004) e Veríssimo e Rodrigues (2000). Na perspectiva da detecção de defeitos em componentes, o serviço de detecção é implementado observando o comportamento dos componentes em caso de falhas. Portanto, a tipificação é realizada a partir da maneira como as falhas afetam o comportamento dos componentes do sistema. As principais classes de falhas, comumente utilizadas na literatura técnica relacionada, são:

- *Falha por parada* (*Crash fault*) – Ocorre quando o componente para de funcionar, deixando de responder a qualquer requisição de serviço. Esse é o tipo de

falha mais comum nos modelos de sistemas relacionados ao projeto de sistemas distribuídos.

- *Falha por omissão* (*Omission fault*) – Ocorre quando o componente omite (ou deixa de produzir) o resultado esperado, conforme sua especificação. Observe que a falha por parada é um tipo particular de falha por omissão, na qual o componente omite permanentemente os resultados esperados.
- *Falha temporal* (*Timing fault*) – Ocorre quando o componente responde fora do prazo especificado. Observe que a falha por omissão poder ser caracterizada como uma falha temporal, na qual o resultado esperado nunca será produzido dentro do prazo, caso este exista.
- *Falha de valor* (*Value fault*) – Ocorre quando o componente produz uma saída incorreta para um dado valor de entrada.
- *Falha arbitrária*, *maliciosa* ou *bizantina* (*Byzantine fault*) – Ocorre quando um componente pode manifestar qualquer tipo de comportamento na presença de falha, podendo parar de funcionar, omitir resultados, produzir resultados fora do prazo ou ainda produzir valores incorretos.

A Figura 2.3 apresenta o relacionamento entre estes diferentes tipos de falhas.

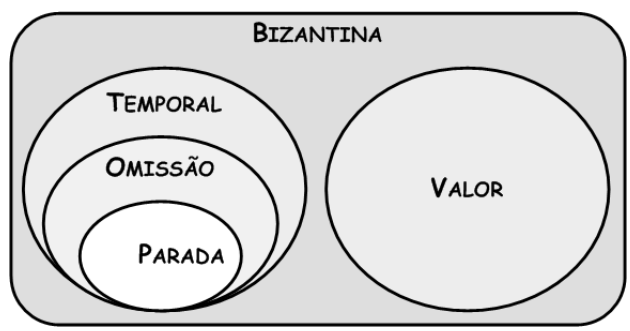


Figura 2.3: Tipificação das falhas de componente

Neste capítulo são discutidos os aspectos relacionados à detecção de defeitos causados por falhas por parada ou *crash*, onde temporizadores são usados como elementos principais na implementação de tais detectores. Falhas por parada são as mais comuns e existem técnicas que podem ser utilizadas para transformar falhas mais severas em falhas por parada (CRISTIAN, 1991).

De um modo geral, o projeto de detectores de defeitos de parada, consiste em determinar valores adequados para os temporizados e em definir de quanto em quanto

tempo as verificações devem ser realizadas – o que pode ser uma tarefa difícil a depender do modelo de sincronia a ser considerado para o sistema distribuído (ver Seção 2.1.4). O tratamento de outros tipos de falhas, como de valor e bizantinas, pode ser realizado a partir de esquemas de temporizadores combinados a técnicas de mascaramento e hierarquia de falhas (CRISTIAN, 1991). Uma discussão mais detalhada sobre a implementação de estratégias de detecção de defeitos de valor e bizantinos podem ser encontradas em Cristian (1991); Doudou e outros (1999); Alvisi e outros (2001); Doudou, Garbinato e Guerraoui (2002); Leroy (2003); Kihlstrom, Moser e Melliar-Smith (2003) etc.

2.1.3 Níveis de confiabilidade

Conhecidos os tipos de comportamentos defeituosos que um componente pode expor e suas respectivas probabilidades (i.e. conhecido o modelo de falhas), diferentes técnicas de suporte à confiabilidade podem ser implementadas. A escolha de determinada técnica se dá conforme o nível de confiabilidade desejado para os diversos componentes do sistema e custos de projeto e operação relacionados.

Os níveis de confiabilidade estão relacionados às diferentes garantias que o sistema pode prover em caso de falhas, sendo estas garantias definidas em termos de duas propriedades básicas: *liveness* e de *safety*.

Segundo Lamport e Lynch (1989), informalmente, a propriedade *safety* assegura que as transições de estados do sistema levam apenas a estados corretos. Isto é, se um resultado de algum processamento é produzido, este atende a especificação do sistema. A propriedade de *liveness* está associada à terminação, ou seja, um estado correto é alcançado em algum momento. Deste modo, pode-se dizer que a propriedade *safety* está relacionada à correção das ações realizadas pelo sistema, enquanto que a propriedade *liveness* está relacionada à capacidade do sistema em se manter operacional (i.e., *live*).

A combinação das propriedades *safety* e *liveness* conferem ao sistema quatro diferentes níveis de confiança em seu funcionamento ou dependabilidade, de acordo com (GÄRTNER, 1999):

- Se o sistema não é capaz de garantir nem *safety* nem *liveness*, então o mesmo é dito não confiável (*unreliable*).
- Se o sistema garante *safety*, mas não garante *liveness*, o mesmo é dito *fail-safe* – isto é, o mesmo não é capaz de continuar operacional, mas realiza uma parada segura (ou desligamento seguro).

- Se o sistema garante *liveness*, mas não garante *safety*, o mesmo continuará operacional, podendo realizar algumas ações incorretas – neste caso, a ocorrência da falha pode ser percebida pelo sistema usuário, mas o sistema pode continuar operando mesmo que de forma degradada. Neste nível de confiabilidade, é possível que ações automáticas de recuperação sejam realizadas de modo a permitir que o sistema volte a produzir resultados corretos.

- Por fim, se o sistema garante *safety* e *liveness*, o mesmo é capaz de mascarar a ocorrência da falha, não permitindo que a mesma seja percebida pelo sistema usuário.

A Figura 2.4 apresenta um diagrama que ilusta os níveis de confiança no funcionamento do sistema de acordo com as propriedades de *safety* e *liveness*.

Figura 2.4: Nívels de confiabilidade

Quanto maior o nível de dependabilidade desejado para um sistema, maior é o custo de sua implementação. Deste modo, um sistema capaz de mascarar a ocorrência de falhas possui um custo de implementação maior que um sistema que garante apenas funcionamento degradado, e assim por diante.

Se a propriedade de *liveness* é especificada considerando aspectos temporais (ou componentes de sincronia) então a mesma é chamada de *timeliness*. Por exemplo, "Uma mensagem enviada é recebida por seu destinatário em no máximo D unidades de tempo".

2.1.4 Modelos de sincronia

Em sistemas distribuídos, a sincronia temporal é uma característica importante na formulação de modelos, sendo assim um aspecto fundamental na implementação dos de-

tectores de defeitos por parada ou *crash*. Os modelos de sistemas distribuídos síncronos e assíncronos delimitam extremos opostos (LAMPORT; LYNCH, 1989). Em um modelo de sistema completamente assíncrono, os limites temporais são inexistentes - não se pode fazer considerações sobre os tempos de processamento e de transmissão das mensagens. Em um sistema síncrono, por outro lado, existem limites temporais para ações de processamento e comunicação.

Os modelos assíncronos são adequados para serem utilizados em ambientes com um alto grau de incerteza temporal. Entretanto, em tais modelos puramente assíncronos, certos problemas básicos encontrados na construção de mecanismos de tolerância a falhas, a exemplo do consenso distribuído, não possuem solução determinística (FISCHER; LYNCH; PATERSON, 1985).

Os modelos de sistemas síncronos são adequados para sistemas em que as restrições temporais são imperativas para o correto funcionamento do sistema. Entretanto, as definições dos limites temporais podem ser bastante difíceis de serem obtidas na prática. Por exemplo, o tempo máximo de execução de um algoritmo depende da implementação da plataforma de hardware, do processo de geração do código objeto do programa, das políticas e facilidades de comunicação usadas, dos defeitos em componentes, da carga computacional imposta pelas aplicações, do grau de dinamicidade e de heterogeneidade do ambiente etc. Além disso, falhas na especificação de limites temporais podem levar os sistemas a apresentarem defeitos durante a execução.

De certo modo, a escolha do modelo entre síncrono ou assíncrono pode não ser a mais adequada para a concepção de muitos sistemas distribuídos. Sistemas reais, em geral, apresentam algum tipo de comportamento síncono em certas partes do sistema ou em certas condições de operação. Esse fato tem motivado o desenvolvimento de modelos que ocupam uma posição intermediária entre os modelos síncronos e assíncronos (DWORK; LYNCH; STOCKMEYER, 1988). Exemplos de tais modelos intermediários são o modelo *timed asynchronous* (CRISTIAN; FETZER, 1999), o assíncrono com detectores de defeitos (CHANDRA; TOUEG, 1996) e o modelo síncrono particionado (MACÊDO; GORENDER, 2009, 2012). Nesses modelos intermediários, em geral existem soluções para problemas básicos de tolerância a falhas, impossíveis de serem resolvidos no modelo asíncrono puro, como o citado consenso distribuído – ver Dwork, Lynch e Stockmeyer (1988); Veríssimo e Casimiro (2002); Gorender e Macêdo (2002); Gorender, Macêdo e Raynal (2005); Veríssimo (2006); Macêdo e Gorender (2009); Macêdo e Gorender (2012) etc.

No modelo híbrido *TCB* (*Timely Computing Base*) Veríssimo e Casimiro (2002), um sub-sistema síncrono interliga todos os elementos do sistema distribuído assíncrono. A proposta do TCB foi generalizada em Veríssimo (2006), o qual propõe o

modelo de *Wormholes*. Tal modelo pressupõe a existência de um modelo assíncrono equipado com componentes com propriedades especiais, dito *wormholes*, que garantem certas propriedades como sincronia ou segurança, as quais permitem que problemas fundamentais de sistemas distribuídos assíncronos possam ser solucionados.

Para atender aos requisitos operacionais de sistemas distribuídos híbridos, porém dinâmicos, outros modelos foram propostos: Gorender e Macêdo (2002); Gorender, Macêdo e Raynal (2005); Macêdo (2007); Gorender, Macêdo e Raynal (2007); Macêdo e Gorender (2009); e Gorender e Macêdo (2010). Estes modelos (e seus respectivos mecanismos básicos subjacentes) visam lidar com os aspectos dinâmicos e híbridos dos sistemas distribuídos modernos, atendendo às demandas dos novos ambientes com QoS variadas.

Em Gorender e Macêdo (2002), foi apresentado um algoritmo de consenso que requer uma *spanning tree* síncrona no sistema distribuído, onde processos são síncronos e canais de comunicação podem ser síncronos ou assíncronos. Nos trabalhos de Gorender, Macêdo e Raynal (2005, 2007), o requisito de *spanning tree* síncrona foi removido e foram apresentadas soluções para o consenso uniforme em ambientes dinâmicos. Em Macêdo (2007), o modelo foi generalizado para que processos e canais de comunicação pudessem variar entre síncrono e assíncrono e foi apresentado em Macêdo (2007) e Macêdo e Freitas (2009) um algoritmo de comunicação em grupo capaz de ligar com ambientes híbridos e dinâmicos.

Em Macêdo e Gorender (2008, 2009) foi introduzido o modelo *partitioned synchronous* que requer menos garantias temporais do que o modelo síncrono e através do qual foi provado ser possível a implementação de detectores perfeitos e conseno distribuído quando falhas por parada de processos são consideradas. Vale salientar que a implementação de detectores perfeitos no modelo *partitioned synchronous* não requer a existência de um *wormhole* síncrono (VERÍSSIMO; CASIMIRO, 2002) ou *spanning tree* síncrona (GORENDER; MACÊDO, 2002), onde seria possível implementar ações síncronas globais em todos os processos, como sincronização interna de relógios. No sistema *partitioned synchronous* proposto, componentes do ambiente distribuído necessitam ser síncronos, mas os mesmos não precisam estar conectados entre si via canais síncronos. E mesmo que parte dos processos não esteja em qualquer das componentes síncronas, pode-se ainda assim tirar algum proveito das partições síncronas existentes para melhorar a robustez das aplicações para tolerar falhas (MACÊDO; GORENDER, 2009). O estudo de soluções de tolerância a falhas para o modelo *partitioned synchronous* tem interesse prático uma vez que muitas configurações reais incluem componentes síncronas, como, por exemplo, dispositivos em NCS locais que se comunicam com dispositivos ou supervisórios em NCS remotas através de redes WAN.

2.2 Detecção de Defeitos em Sistemas Distribuídos

No contexto dos sistemas distribuídos, o detector de defeitos é um serviço com módulos de software distribuídos nos dispositivos que fazem parte do sistema (CHANDRA; TOUEG, 1996). Esses módulos se comunicam com o intuito de trocar informações a respeito dos estados dos nós que estão sendo monitorados. A depender do estilo de monitoramento usado, essa troca de mensagens pode implicar em maiores ou menores custos computacionais (FELBER, 1998).

Na detecção de defeitos por *crash*, os módulos monitores do detector de defeitos usam temporizadores para determinar prazos (i.e. *timeouts*) para chegada das mensagens de monitoramento oriundas dos nós monitorados – o não atendimento destes prazos sinaliza possíveis falhas dos nós monitorados.

Em sistemas distribuídos síncronos, uma vez que os limites temporais para o processamento e transmissão das mensagens são conhecidos, os *timeouts* podem ser estabelecidos com precisão – assim, quando um nó monitorado não atende ao prazo especificado, o mesmo é tido como falho. Em sistemas assíncronos (ou parcialmente sincrônos), por outro lado, os limites temporais para o processamento e transmissão das mensagens são desconhecidos e podem variar de forma arbitrária. Por conta disso, a definição de *timeouts* é um problema e não se pode determinar com certeza se um nó monitorado efetivamente falhou ou se a mensagem de monitoramento está atrasada (FISCHER; LYNCH; PATERSON, 1985). Nesses casos,, quando uma mensagem não chega no prazo, o nó monitorado é apenas suspeito de ter falhado (CHANDRA; TOUEG, 1996). Para reduzir o número de falsas suspeitas, *timeouts* mais longos podem ser usados, o que implica em longas latências de detecção, que por sua vêz pode comprometer o desempenho das aplicações ou dos demais mecanismos de tolerância a falhas que dependem do detector.

Na tentativa de evitar o uso de *timeouts* muito longos (o que é prejudicial para a velocidade das detecções), reduzindo o número de falsas suspeitas, estratégias de adaptação de *timeout* vêm sendo usadas – ver, por exemplo: Macêdo (2000); Chen, Toueg e Aguilera (2002); Bertier, Marin e Sens (2002); Macêdo e Lima (2004); Nunes e Jansch-Pôrto (2004); Lima e Macêdo (2005); Sá e Macêdo (2006) etc. Estas estratégias utilizam estimadores de atrasos, os quais se baseiam em alguma informação da comunicação entre nós para realizar suas estimativas.

Como já mencionado, detectores de defeitos em sistemas assíncronos são não confiáveis, pois podem apontar a falha de um nó correto (i.e. em funcionamento), ou deixar de apontar a falha de um nó efetivamente defeituoso (CHANDRA; TOUEG, 1996).

Esses detectores podem ser classificados de acordo com a sua capacidade de detectar defeitos (i.e. correção) e de evitar falsas suspeitas (i.e. precisão). A classificação é importante para determinar o potencial dos detectores de defeitos na resolução de problemas fundamentais de tolerância a falhas em sistemas distribuídos (RAYNAL, 2005) – sendo usada para auxiliar na composição de modelos de sistemas de distribuídos e ajudar na prova das propriedades de *safety* e *liveness* dos algoritmos de tolerância a falhas. Todavia, do ponto de vista operacional, a classificação dos detectores baseada na correção e precisão, não é suficiente para permitir aspectos de desempenho sejam avaliados, tais como: velocidade e confiabilidade do detector. Com isso, métricas de QoS são usadas no projeto e avaliação dos detectores de defeitos de modo a permitir uma adequação dos mesmos aos requisitos das aplicações (CHEN; TOUEG; AGUILERA, 2002).

As subseções a seguir abordam questões de implementação da detecção de defeitos.

2.2.1 Estilos de Monitoramento de Defeitos

Em um ambiente distribuído, a implementação de detectores de defeitos considera dois estilos básicos para o monitoramento remoto do estado dos nós do sistema: *Pull* e *Push* (FELBER, 1998).

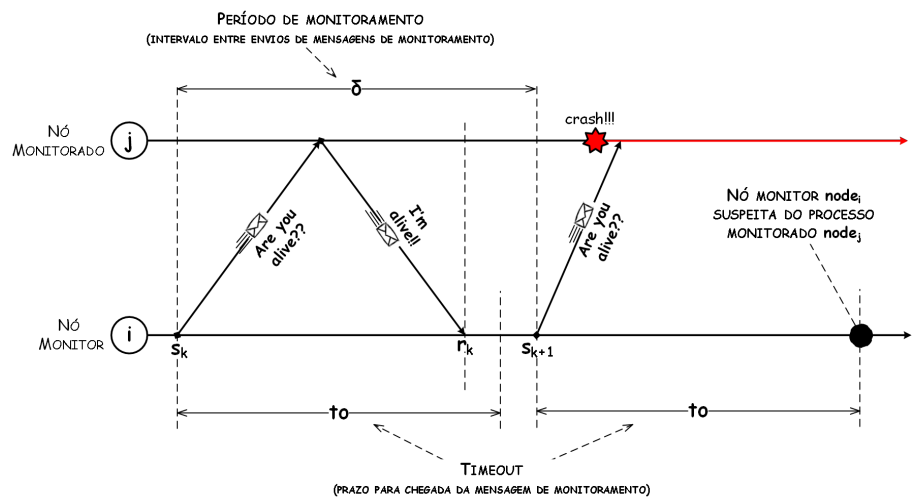


Figura 2.5: Estilo de monitoramento Pull

No estilo de monitoramento *Pull*, Figura 2.5, o nó monitor (p_i) do detector de defeitos[1] envia periodicamente uma mensagem de monitoramento do tipo *"Are you Alive?"*,

[1]Note que, na discussão apresentada, o nó representa tanto o dispositivo quanto o programa em execução,

dita aya. Uma vez recebida a mensagem de *"Are you Alive?"*, o nó monitorado (p_j) deve responder com seu atual estado usando uma mensagem do tipo *"I am alive!"* ou *heartbeat*, dita hb. A cada mensagem de monitoramento enviada, o nó monitor deve estimar o intervalo de tempo necessário (*timeout*) para a chegada da mensagem de resposta oriunda do nó monitorado. Caso a mensagem não chegue dentro do intervalo esperado, o nó monitor passa a suspeitar da falha do nó monitorado. Neste estilo de monitoramento, o nó monitor controla o ritmo do monitoramento e calcula os timeouts baseados nos atrasos de ida-e-volta (*round-trip-time*). Sendo assim, uma abordagem de detecção, construída a partir de tal estilo de monitoramento, consegue ter uma estimativa mais eficiente dos atrasos de comunicação e também controlar de forma mais eficiente o ritmo (i.e. período) de monitoramento.

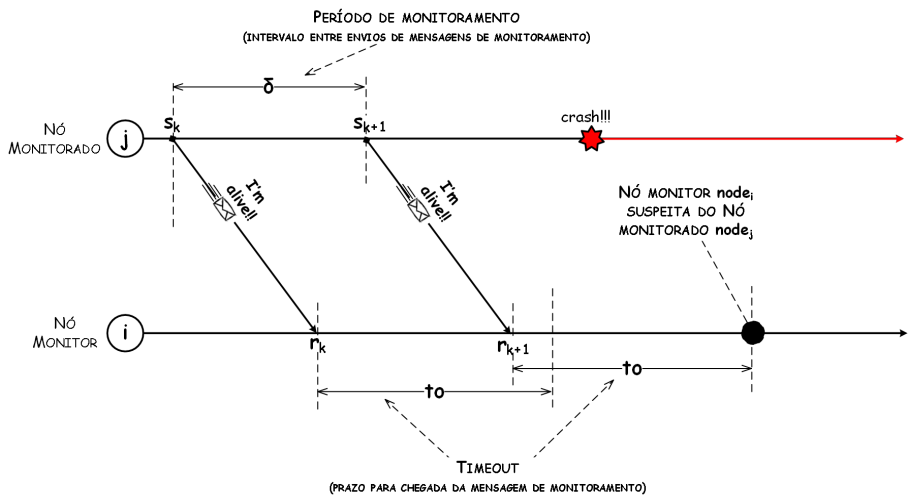


Figura 2.6: Estilo de monitoramento Push

No estilo *Push*, Figura 2.6, o nó monitorado (p_j) espontaneamente envia seu estado atual para o nó monitor (p_i). Baseado no intervalo entre chegada das mensagens, o nó monitor deve estimar o instante de chegada da próxima mensagem de monitoramento. Caso a mensagem não chegue dentro do intervalo esperado, o nó monitor suspeita da falha do nó monitorado. Neste estilo de monitoramento, o número de mensagens de monitoramento trocadas entre os módulos monitor e monitorado é menor que no estilo *Pull*, o que implica em um menor uso do canal de comunicação.

onde o módulo do detector de defeitos é uma das funções embutidas neste programa. Portanto, assume-se que a falha do programa representa a falha do próprio dispositivo.

2.2.2 Classes de Detectores de Defeitos

O serviço de detecção de defeitos deve manter uma lista contendo os nós considerados corretos (i.e., não suspeitos de falhas) e falhos (ou suspeitos de terem falhado). A capacidade do detector de defeitos em manter esta lista com informações prontamente atualizadas depende do modelo de sincronia considerado. Em sistemas síncronos é possível implementar detectores capazes de prover informações confiáveis, mas o mesmo não se pode afirmar para sistemas assíncronos (i.e., *time-free* ou livres de tempo). Entretanto, sistemas reais não são necessariamente assíncronos, possuindo algum nível de sincronia (ver Seção 2.1.4). Nesse contexto, Chandra e Toueg (1996) propõem o uso de detectores de defeitos não confiáveis como uma estratégia para ocultar a sincronia necessária à resolução de problemas de tolerância a falhas em sistemas distribuídos assíncronos, sem a necessidade explícita de embutir as hipóteses de sincronia nos algoritmos e protocolos que fazem uso desses detectores de defeitos. O argumento de Chandra e Toueg (1996) se baseia no fato de que sistemas reais apresentam períodos de estabilidade (GST[2]), nos quais os atrasos de processamento e troca de mensagens são desconhecidos, mas limitados – o que permite que um teste temporal possa em algum momento detectar, de forma definitiva, a falha por *crash*. Assim, é possível embutir essa hipótese de sincronia no detector de defeitos, sem a necessidade de a mesma ser considerada na implementação dos algoritmos – bastando, portanto, que os algoritmos acreditem que em algum momento o detector de defeitos apontará como defeituoso um nó que tenha falhado e deixará de apontar, como falho, um nó correto.

A fim de agrupar as garantias providas por diferentes implementações de detectores de defeitos com suas respectivas capacidades de resolução de problemas em sistemas distribuídos, Chandra e Toueg (1996) propõem uma classificação para os detectores de defeitos baseada nas propriedades de Completude (*Completeness*) e Precisão (*Accuracy*).

A Completude determina a capacidade dos detectores de defeitos em identificar os nós defeituosos. De acordo com esta propriedade, o detector pode apresentar uma das seguintes características: (a) ***strong completeness*** – todos os nós defeituosos são permanentemente suspeitos de falha por todos os nós corretos; e (b) ***weak completeness*** – todos os nós defeituosos são permanentemente suspeitos de falha por algum nó correto.

A propriedade de Precisão, por sua vez, determina a capacidade do detector em evitar erros de detecção (i.e., falsas suspeitas). De acordo com esta propriedade, o detector pode apresentar uma das seguintes características: (a) **strong accuracy** – nenhum nó correto é suspeito de falha; (b) **weak accuracy** – algum nó correto nunca é suspeito

[2] *Global Stabilization Time* (DWORK; LYNCH; STOCKMEYER, 1988).

de falha; (c) **eventual strong accuracy** – existe um tempo depois do qual nenhum nó correto é suspeito de falha; e (d) **eventual weak accuracy** – existe um tempo depois do qual algum nó correto deixa de ser suspeito de falha.

Oito classes de detectores são definidas a partir da Completude e da Precisão, conforme Tabela 2.1.

Tabela 2.1: Classes de detectores de defeitos

Completeness	Accuracy			
	Strong	Weak	Eventually Strong	Eventually Weak
Strong	$\mathcal{P}$	$\mathcal{S}$	$\diamond\mathcal{P}$	$\diamond\mathcal{S}$
Weak	$\mathcal{D}$	$\mathcal{W}$	$\diamond\mathcal{D}$	$\diamond\mathcal{W}$

Dentre as diferentes classes, os detectores das classes $\mathcal{P}$ e $\diamond\mathcal{W}$ são os que apresentam o maior e o menor nível de garantias (ou qualidade de serviço) em termo das propriedades de Completude e Precisão, respectivamente (CHANDRA; TOUEG, 1996).

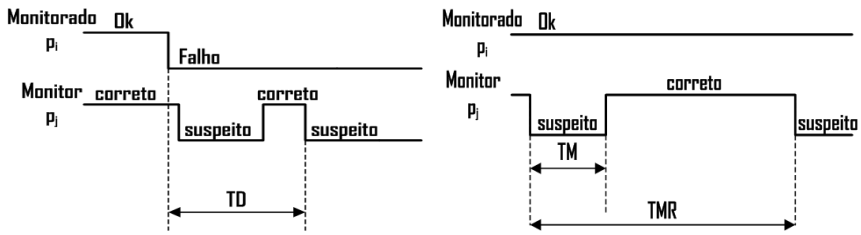
2.2.3 Qualidade de Serviço em Detecção de Defeitos

A classificação proposta por Chandra e Toueg (1996) é importante para definir que garantias invariantes um detector de defeitos pode prover, a depender do modelo de sistema distribuído considerado. A implementação de um detector da classe $\mathcal{P}$, por exemplo, é impossível em modelos de sistemas distribuídos assíncronos. Por outro lado, os detectores da classe $\diamond\mathcal{S}$ são os detectores de defeitos que encapsulam o menor grau de sincronia para resolução do problema do consenso em sistemas distribuídos parcialmente síncronos (ou assíncronos equipados com detectores de defeitos).

Por outro lado, a classificação proposta por Chandra e Toueg (1996) é baseada em comportamentos temporais futuros não quantificáveis, sendo, portanto, não apropriada para algumas aplicações que necessitem que a qualidade de serviço dos detectores de defeitos seja expressa de forma mais explícita. A classificação de Chandra e Toueg (1996), não permite, por exemplo, que duas implementações da mesma classe de detetor possam ser comparadas em termos de desempenho. Para contornar essa dificuldade, Chen, Toueg e Aguilera (2002) propuzeram métricas probabilísticas de QoS para medir a capacidade do detector em prevenir falsas suspeitas (*precisão*) e a rapidez com a qual o mesmo detecta defeitos. Estas métricas estão divididas em **Métricas Primárias** e **Métricas Secundárias**.

Métricas Primárias de QoS de Detecção. São métricas que não podem ser deduzidas de quaisquer outras métricas, mas a partir das quais se deduz as demais. As

seguintes métricas são definidas como primárias: **Tempo de detecção** (TD, *Detection Time*), intervalo de tempo necessário para que um nó monitor ($node_j$) suspeite a falha de um nó monitorado ($node_i$) – é medido a partir do instante no qual o nó monitorado efetivamente falhou (ver Figura 2.7(a)); **Intervalo entre falsas suspeitas** (TMR, *Mistake Recurrence Time*), intervalo de tempo entre duas falsas suspeitas de falhas consecutivas (ver Figura 2.7(b)); e **Duração das falsas suspeitas** (TM, *Mistake duration*), intervalo de tempo que o detector leva para corrigir uma falsa suspeita de falha (ver Figura 2.7(b)).



(a) Tempo de Detecção (TD)

(b) Duração da Falsa Suspeita (TM) e Intervalo Entre Falsas Suspeitas (TMR)

Figura 2.7: Métricas Primárias de QoS

Métricas Secundárias de QoS de Detecção. São métricas derivadas das métricas primárias. As seguintes métricas são definidas como secundárias: **Taxa de falsas suspeitas** (λ_M, *Average Mistake Rate*) – indica a taxa de falsas suspeitas de falhas cometidas pelo detector; **Probabilidade de consulta correta** (PA, *Query Accuracy Probability*) – indica a probabilidade do detector produzir a saída correta em um instante qualquer; **Período bom** (TG, *Good Period Duration*) – indica o intervalo médio de tempo em que o detector de defeitos produz a saída correta; e **Intervalos entre períodos bons** (TFG, *Foward Good Period Duration*) – indica o intervalo médio de tempo entre dois períodos bons TG.

Relação entre as Métricas de *QoS* de Detecção. Sejam $Pr(A)$, $E(x)$, $E(x^{(k)})$, $V(x)$ a probabilidade de um evento A ocorrer, o valor esperado de x, o k-ésimo momento e a variância de x, respectivamente, as relações entre as métricas seguem o teorema a seguir (CHEN; TOUEG; AGUILERA, 2002):

Teorema 2.2.1 (Relação entre as métricas de *QoS* de Detecção de Defeitos) *Para qualquer detector de defeitos que lentamente esquece o histórico de suas saídas em rodadas livres de defeitos, os seguintes resultados se mantêm:*

1. $TG = TMR - TM$.
2. *Se* $0 < E(TMR) < \infty$, *então* $\lambda_M = \dfrac{1}{E(TMR)}$ *e* $PA = \dfrac{E(TG)}{E(TMR)}$.
3. *Se* $0 < E(TMR) < \infty$ *e* $E(TG) = 0$, *então* $TFG = 0$. *Se* $0 < E(TMR) < \infty$ *e* $E(TG) \neq 0$, *então:*
   (a) $\forall x \in [0, \infty)$, $Pr(TFG \leq x)$, $\dfrac{1}{E(TG)} \displaystyle\int_0^x Pr(TG > y)dy$.
   (b) $E(TG^{(k)}) = \dfrac{E(TG^{(k+1)})}{[(k+1) * E(TG)]}$, *em particular* $E(TFG) = \left[\dfrac{1 + V(TG)}{E(TG)^2}\right] * \dfrac{E(TG)}{2}$.

Com base no Teorema 2.2.1, algumas conclusões podem ser obtidas, por exemplo: (a) os bons períodos acontecem nos intervalos em que o detector não comete falsas suspeitas; (b) a taxa de falsas suspeitas do detector de defeitos representa o número de falsas suspeitas por unidade de tempo; (c) a probabilidade do detector produzir um valor correto é a relação entre o valor esperado para períodos bons e o valor esperado para intervalo entre falsas suspeitas; e (d) se existe a possibilidade do detector não cometer erros, então o valor esperado para o próximo período bom será a relação entre o somatório das probabilidades de todos os possíveis intervalos de períodos bons e valor esperado para o período bom.

Sintonia de Detectores de Defeitos com QoS. Chen, Toueg e Aguilera (2002) propuzeram um processo de sintonia (*off-line*) para determinar, através do comportamento probabilístico do canal de comunicação, o período de monitoramento (Δ) do detector e a margem de segurança (SM). SM compõe o *timeout* detecção e é usada para evitar que atrasos extras levem o detector a cometer falsas suspeitas. Esta abordagem de sintonia do detector de defeitos está sujeita ao nível de QoS desejado, o qual é especificado a partir da tupla $\langle TD_U, TMR_L, TM_U \rangle$ definida pelo usuário (ou pela aplicação). As seguintes relações devem ser atendidas:

$$TD < TD_U,\ E(TMR) \geq TMR_L,\ E(TM) \leq TM_U \tag{2.1}$$

O processo de sintonia é um problema de pesquisa operacional, no qual se deve encontrar o máximo Δ sujeito às restrições da Equação 2.1. Para tanto, de posse de Δ e TD_U, pode-se encontrar uma margem de segurança inicial $SM^{(0)}$, através do procedimento de configuração a seguir.

(a) Calcular $q^{(0)} = (1 - p_L) * Pr(D < TD_U)$, em que p_L e $q^{(0)}$ representam, respectivamente, a probabilidade de perda de mensagens e a probabilidade de uma mensagem de monitoramento $m^{(k)}$ chegar dentro do intervalo $[k, k + 1)$. Com isso, é possível calcular o período de monitoramento máximo, usando $\Delta_{max} = q^{(0)} * TM_U$. Se $\Delta_{max} = 0$, então a *QoS* desejada não pode ser encontrada, senão o passo **(b)** deve ser realizado.

(b) Encontrar o maior $\Delta < \Delta_{max}$ tal que $f(\Delta) \geq TMR_L$, em que:

$$f(\Delta) = \Delta * \left\{ q^{(0)} * \prod_{j=1}^{\left\lceil \frac{TD_U}{\Delta} - 1 \right\rceil} [p_L + (1 + p_L) * Pr(D > TD_U - j * \Delta)] \right\}^{-1} \tag{2.2}$$

(c) Por fim, encontra-se $SM^{(0)} = TD_U - \Delta$

O procedimento de sintonia propostos por Chen, Toueg e Aguilera (2002) são realizados em tempo de projeto, o que pode representar um problema quando as características do ambiente distribuído mudam durante a execução. Muitos ambientes distribuídos dinâmicos (e.g., CPS) requerem detectores de defeitos com habilidade de auto-sintonia. Estes detectores auto-sintonizáveis são chamados de detectores autonômicos e sua implementação é discutida com detalhes na Seção 3.3.

2.3 Detecção Adaptativa de Defeitos

Mediante a natureza arbitrária dos atrasos de processamento e de comunicação nos sistemas assíncronos, a definição do período de monitoramento e do *timeout* de detecção representa um desafio. O período de monitoramento impacta no custo computacional, na velocidade e na precisão da detecção, enquanto que o *timeout* de detecção impacta na velocidade e na precisão do detector. Muitos trabalhos têm lidado com a escolha do período de monitoramento em tempo de projeto, enquanto que lida com a qualidade do serviço de detecção através da adaptação do timeout de detecção – permitindo a provisão de uma detecção mais precisa e com um menor tempo de detecção.

Essa adaptação de *timeout* se apoia na mesma premissa de Chandra e Toueg (1996) de que sistemas reais experimentam períodos de estabilidade – i.e., o GST, *Global Time Stabilization* (DWORK; LYNCH; STOCKMEYER, 1988). A maioria das estratégias de adaptação, usadas na implementação dos detectores adaptativos de defeitos, considera que, a partir de alguma informação a respeito do ambiente ou das mensagens de

monitoramento do serviço de detecção, é possível realizar estimativas a respeito dos atrasos e prover *timeouts* de detecção adequados – ver por exemplo: Bertier, Marin e Sens (2002); Macêdo e Lima (2004); Nunes e Jansch-Pôrto (2004); Falai e Bondavalli (2005); Wiesmann, Urban e Defago (2006) etc. Essas hipóteses, apesar de serem mais restritivas que as observadas nos detectores de Chandra e Toueg (1996), são geralmente verificadas na prática – ver por exemplo, Jain e Routhier (1986), Afanasyev e outros (2010) etc.

Em termos de implementação, a adaptabilidade do detector de defeitos consiste em estimar *timeouts* de detecção, considerando algum estimador de atrasos em conjunto com uma margem de segurança – definida estática ou dinamicamente e cujo papel é evitar que valores de atraso subestimados leve a falsas suspeitas de falhas. Algumas estratégias de detecção adaptativa de defeitos em sistemas distribuídos de automação são discutidas nas Seções 3.1 e 3.2. A seguir são discutidos aspectos mais gerais sobre a detecção adaptativa e apresentados exemplos de abordagens clássicas usadas em sistemas distribuídos.

2.3.1 Aspectos Gerais sobre Adaptação do *Timeout* de Detecção

As estratégias de adaptação de timeout são independentes do estilo de monitoramento – isto é, um mesmo algoritmo pode ser usado tanto no estilo de monitoramento *Pull* quanto no estilo *Push* (com nenhuma ou apenas com algumas pequenas adaptações).

Para facilitar a discussão sobre a adaptação do *timeout*, permitindo que questões relacionadas ao estilo de monitoramento não precisem ser consideradas, denotaremos o atraso medido por D. Assim, no estilo de monitoramento *Push*, D é estimado pelo monitor a partir do intervalo entre chegadas das mensagens de *heartbeat*, isto é $D^{(k)} = t_a^{(k)} - t_a^{(k-1)}$, em que $t_a^{(k)}$ representa instante de chegada do k-ésimo heartbeat ($hb^{(k)}$), enquanto que $D^{(k)}$ representa o (k)-ésimo intervalo entre chegadas dos heartbeats $hb^{(k)}$ e $hb^{(k-1)}$. Por outro lado, no estilo de monitoramento *Pull*, D é estimado pelo monitor a partir do atraso de ida-e-volta da mensagem de monitoramento, isto é $D^{(k)} = t_a^{(k)} - t_s^{(k)}$, em que $D^{(k)}$ representa o k-ésimo atraso de ida-e-volta medido e $t_s^{(k)}$ e $t_a^{(k)}$ representam, respectivamente, o instante de envio do k-ésimo are "you alive?" ($aya^{(k)}$) e o instante do recebimento do k-ésimo heartbeat ($hb^{(k)}$).

Para a adaptação do *timeout* de detecção, as estratégias usadas na construção dos detectores adaptativos calculam uma estimativa EST para o próximo atraso (D), a partir de um conjunto de observações a respeito do ambiente de execução, por exemplo histórico de atrasos, perda de mensagens etc. Além disso, estas estratégias também consideram a estimativa (*offline* ou *online*) de uma margem de segurança SM, baseada

nas variações do atraso e utilizada para evitar falsas suspeitas de falhas. Assim, o k-ésimo *timeout* de detecção ($TO^{(k)}$) pode ser estimado:

$$TO^{(k)} = \beta * EST^{(k)} + \gamma * SM^{(k)} \tag{2.3}$$

em que β e γ representam os fatores de confiança atribuídos a estimativa do atraso e a estimativa da margem de segurança, respectivamente.

2.3.2 Estratégias para Adaptação do *Timeout* de Detecção

Na literatura relacionada à detecção adaptativa de defeitos, existem inúmeras estratégias propostas para a adaptação do timeout de detecção, por exemplo: Macêdo (2000), Nunes e Jansch-Pôrto (2004), Macêdo e Lima (2004), Sá e Macêdo (2006), Bertier, Marin e Sens (2002), Chen, Toueg e Aguilera (2002), Falai e Bondavalli (2005) etc. Entretanto, nesta subseção foram selecionadas apenas algumas daquelas que foram avaliadas na implementação de sistemas distribuídos de automação, conforme apresentado Seção 3.1. A seguir, são discutidas algumas das estratégias de adaptação de *timeout* selecionadas.

Estratégia de Adaptação de Timeout de Chen, Toueg e Aguilera (2002). Nesta estratégia, realiza-se a estimativa do atraso a partir da média dos atrasos relacionados aos W últimos heartbeats recebidos:

$$EST^{(k)} = \frac{1}{W} * \left(\sum_{i=k-W+1}^{k} D^{(i)} \right) \tag{2.4}$$

Então, o *timeout* de detecção é calculado usando uma margem de segurança constante:

$$TO^{(k)} = EST^{(k)} + SM \tag{2.5}$$

A margem de segurança (SM) é obtida *offline* a partir dos requisitos de qualidade de serviço definidos e de hipóteses sobre o comportamento probabilístico dos atrasos (D).

Estratégia de Adaptação de Timeout de Jacobson (1988). Esta estratégia considera um estimador ARX[3] para estimativa do atraso, conforme apresentado a seguir:

[3] ARX (do inglês, **A**uto**R**egressive e**X**ogeneous) – ver Aguirre (2007).

$$EST^{(k+1)} = \mu * D^{(k)} + (1 - \mu) * EST^{(k)} \tag{2.6}$$

em que μ pondera os efeitos dos valores medidos e estimados no cálculo da próxima estimativa.

Para calcular a margem de segurança, Jacobson (1988) utiliza os desvios entre os valores observados e estimados para o atraso, isto é:

$$SM^{(k+1)} = \mu * (|D^{(k)} - EST^{(k)}|) + (1 - \mu) * SM^{(k)} \tag{2.7}$$

Então, o *timeout* de detecção é calculado da seguinte forma:

$$TO^{(k+1)} = \beta * EST^{(k+1)} + \gamma * SM^{(k+1)} \tag{2.8}$$

Em seu algoritmo original, Jacobson (1988) considera $\mu = \frac{1}{10}$, $\beta = 1$ e $\gamma = 2$.

Estratégia de Adaptação de Timeout de Bertier, Marin e Sens (2002). Esta abordagem usa a estratégia de Chen, Toueg e Aguilera (2002) para a estimativa do atraso. Entretanto, utiliza o algoritmo de Jacobson (1988) sobre o desvio $DSV^{(k)} = D^{(k)} - EST^{(k)}$ para estimar uma margem de segurança dinâmica, isto é:

$$DEST^{(k+1)} = \mu * DSV^{(k)} + (1 - \mu) * DEST^{(k)} \tag{2.9}$$

$$DVAR^{(k+1)} = \mu * (|DSV^{(k)} - DEST^{(k)}|) + (1 - \mu) * DVAR^{(k)} \tag{2.10}$$

$$SM^{(k+1)} = \beta * DEST^{(k+1)} + \gamma * VAR^{(k+1)} \tag{2.11}$$

em que $DEST$ e $DVAR$ são, respectivamente, o desvio estimado e a variação estimada para o mesmo.

Então, Bertier, Marin e Sens (2002) calculam o *timeout* de detecção:

$$TO^{(k+1)} = EST^{(k+1)} + SM^{(k+1)} \tag{2.12}$$

Estratégia de Adaptação de Timeout de Sá e Macêdo (2006). Esta estratégia utiliza uma RNA (Rede Neural Artificial) (HAYKIN, 1994) para aproximar uma função f que sugira valores adequados para o *timeout* de detecção, considerando como entrada o atraso D, a variáção deste atraso $J^{(k)} = D^{(k)} - D^{(k-1)}$ e do período de monitoramento (Δ), isto é:

$$TO^{(k+1)} = f(D^{(k)}, J^{(k)}, \Delta) \tag{2.13}$$

A estratégia utiliza uma RNA do tipo MLP (MultiLayer Pecerptron) estruturada em quatro camadas: uma camada de entrada com três neurônios – um para cada parâmetro de entrada da função f; duas camadas intermediarias com trinta e dez neurônios, respectivamente; e uma camada de saída com apenas um neurônio – para sugerir o valor do timeout de detecção. A RNA adotada é treinada *off-line* usando um algoritmo que sugere taxas de aprendizado adaptativas.

CAPÍTULO 3

Detectores Adaptativos de Defeitos para Ambientes Distribuídos e de Automação

Alirio S. de Sá, Raimundo J. A. Macêdo, Eduardo Cambruzzi e Jean-Marie Farines

Neste Capítulo, são discutidas diferentes implementações de detectores de defeitos adequados a ambientes modernos de automação. Inicialmente, na Seção 3.1, é apresentada a avaliação do impacto da implementação dos detectores de defeitos em Sistemas de Controle sobre Rede. Na Seção 3.2 é discutida a implementação de detectores de defeitos em aplicações distribuídas sobre Redes Veiculares. Por fim, na Seção 3.3, são apresentados detectores auto-gerenciáveis (ou autonômicos), e sua relação com a implementação de plataforma confiáveis em Sistemas Ciber-físicos. Uma discussão sobre os conceitos básicos relacionados à implementação do serviço de detecção de defeitos em ambientes distribuídos foi apresentada no Capítulo 2.

3.1 Detectores Adaptativos de Defeitos para NCS

Um projeto adequado de detectores de defeitos para Sistemas de Controle via Rede (NCS, ver Capítulo 1) deve considerar características inerentes às cargas impostas pelas tarefas do sistema e o modelo de rede de comunicação adotado. Por conta disto, a Seção 3.1.1 apresenta uma descrição geral do exemplo de sistema usado nas discus-

sões sobre os aspectos de projeto dos detectores de defeitos. Os aspectos inerentes ao desempenho do serviço de detecção e do laço de controle são analisados, observando: (i) a QoS de detecção (ver Seção 2.2.3) entregue pelos detectores defeitos em NCS; e (ii) o impacto destes detectores na Qualidade do Controle (QoC, ver Seção 1.3). Estes aspectos de desempenho do detector e do controle são discutidos nas seções 3.1.2 e 3.1.3, respectivamente.

3.1.1 Descrição Geral do Sistema

O NCS usado como exemplo é simulado no TrueTime/Simulink/MatLab – ver Henriksson e Cervin (2007), Mathworks (2006) e Mathworks (2002) para maiores detalhes sobre este simulador.

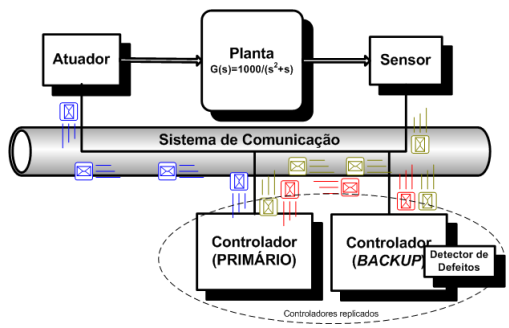


Figura 3.1: NCS com Mecanismos de Tolerância a Falhas

No ambiente de simulação considerado, cada sistema de controle é composto por quatro dispositivos (ver Figura 3.1): um sensor, um atuador e dois controladores replicados, um primário e outro secundário, para tolerar falhas por *crash* de controlador. Esees dispositivos são dotados por sistema operacional multitarefa e de tempo real e interagem entre si via troca de mensagens a partir da rede de comunicação.

No sensor é executada uma tarefa periódica de aquisição de dados (τ_s). A cada amostra coletada por τ_s, uma mensagem com a amostra é enviada para o controlador principal. No controlador, a tarefa de controle (τ_c) é esporádica e é acionada a cada mensagem recebida do sensor. Esta tarefa executa um algoritmo que garante a consistência do estado do controlodor primário com o controlador secundário e envia a ação de controle ao atuador. Quando o atuador recebe a mensagem do controlador, este aciona uma tarefa de atuação (τ_a) responsável por efetivar a ação de controle na planta.

O controlador secundário (ou *backup*) possui um módulo monitor do detector de defeitos embarcado para verificar falhas no controlador primário. O monitoramento

do estado do controlador primário usa o modelo *Push*. Assim, a cada Δ unidades de tempo um *heartbeat* é enviado do controlador primário ao controlador secundário.

Neste ambiente, a rede local garante que as mensagens são entregues em ordem, não existe a possibilidade de partições na rede e, em altas condições de tráfego, podem ocorrer perda de mensagens. A troca de mensagens é implementada sobre uma rede de comunicação *Shared Bus Ethernet* (TANENBAUM, 2003), o que permite que a QoS de detecção e o impacto na QoC possam ser visualizados mais facilmente[1] – alguns detalhes sobre o funcionamento desta rede são discutidos a seguir.

Funcionamento da Rede de Comunicação Shared-Bus Ethernet. Na *Shared-Bus-Ethernet*, antes de enviar uma mensagem, os dispositivos verificam se o meio está livre e, em caso afirmativo, iniciam a transmissão. Quando dois ou mais dispositivos, interessados em enviar uma mensagem, encontram o meio de comunicação disponível e, simultaneamente, iniciam uma transmissão, ocorre uma colisão. Uma vez detectada a colisão, os dispositivos cessam imediatamente suas transmissões e utilizam um protocolo para solucionar a disputa pelo acesso ao meio. Neste protocolo, dito *Recuo Binário Exponencial* (*BEB*,*Binary Exponential Backoff*[2]), após uma colisão, cada dispositivo sorteia um número aleatório de *slots* de tempo e, então, tenta uma nova transmissão após decorrido o número de *slots* de tempo sorteado (TANENBAUM, 2003). Se uma nova colisão acontece, o número de slots de tempo é incrementado e uma nova tentativa é realizada. Esse procedimento se repetirá até que um dispositivo consiga acessar o meio de comunicação ou até que o número máximo de tentativas se esgote e o dispositivo cancele a transmissão. Isso implica, portanto, em possíveis perdas de mensagens. O procedimento *BEB* faz com que o atraso em uma rede *Shared-Bus-Ethernet* tenha um comportamento não determinístico.

Parâmetros do Sistema. No sistema, a tarefa de controle implementa uma lei de controle PID (Proporcional-Integral-Derivativo[3]), com termo integral $T_i = 1$, termo derivativo $T_d = 0,035$, termo proporcional $K = 1,5$ e período de amostragem $h = 10ms$. Esta tarefa é usada para controlar um motor *DC-Servo*, simulado com a seguinte função de transferência:

$$G(s) = \frac{1000}{s * (s + 1)}$$

[1] Uma discussão mais detalhada sobre a implementação de detectores adaptativos de defeitos em NCS, considerando redes de comunicação CAN e *Switched Ethernet*, pode ser encontrada em Sá e Macêdo (2006).

[2] ver Tanenbaum (2003) para maiores detalhes.

[3] ver Ogata (2009) para maiores detalhes.

Por sua vez, o serviço de detecção de defeitos usa períodos de monitoramento $\Delta = \frac{1}{10}, \frac{5}{10}, \frac{9}{10}, 1, 2$ e $5ms$. Além disso, a rede *Shared-Bus-Ethernet* é configurada com uma taxa de transferência de $10Mbps$ e as mensagens enviadas através da rede possuem um tamanho mínimo de 64 *bytes*.

As tarefas do sistema são apresentadas na Tabela 3.1, considerando: tipo de ativação, deadlines, WCET (*Worst-Case Execution Time*) e períodos de ativação.

Tabela 3.1: Especificação das tarefas do sistema

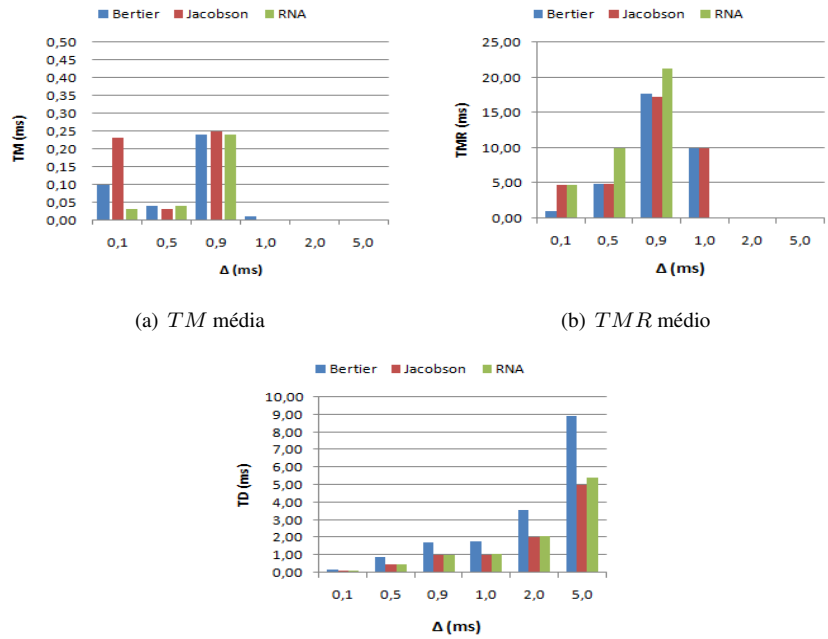
Tarefa	**Dispositivo**	**Ativação**	**Evento**	**Deadline**	**Período**	**WCET**
Controle (τ_c)	Controlador	Esporádica	Recepção da mensagem enviada por τ_s	$6ms$	-	$0,5ms$
Aquisição (τ_s)	Sensor	Periódica	-	$10ms$	$10ms$	$0,4ms$
Atuação (τ_a)	Atuador	Esporádica	Recepção da mensagem enviada por τ_c	$4ms$	-	$0,5ms$
Emissor de *heartbearts*	Controlador Primário	Periódica	-	Δ	Δ	$0.005ms$
Monitor de *heartbeats*	Controlador Secundário	Esporádica	Recepção de um *heartbeat*	Δ	-	$0,005ms$

3.1.2 Qualidade do Serviço da Detecção Adaptativa de Defeitos em Sistema de Controle via Rede

Para verificar a qualidade de serviço da detecção de defeitos entregue ao Sistema de Controle via Rede, são selecionadas as estratégias de detecção adaptativas de Bertier, Marin e Sens (2002), de Jacobson (1988) e de Sá e Macêdo (2007) – ver Seção 2.3.2. O desempenho destas abordagens é verificado em termos dos valores médios das métricas primárias de QoS de detecção (TD, TM e TMR – ver Seção 2.2.3), conforme apresentado na Figura 3.2.

Observando os gráficos das Figuras 3.2(a) a 3.2(c), é possível deduzir que a escolha da estratégia de adaptação de timeouts implica na entrega de diferentes níveis de QoS de detecção. Por exemplo, as abordagens de Jacobson (1988) e de Sá e Macêdo (2007) entregam detecções mais rápidas (i.e., menores tempos de detecção), considerando os diferentes períodos de monitoramento selecionados. Por outro lado, as abordagens de Bertier, Marin e Sens (2002) e Sá e Macêdo (2007) cometem menos falsas suspeitas, além de, na maioria dos casos, corrigirem suas falsas suspeitas mais rapidamente que a abordagem de Jacobson (1988) nestes mesmos cenários.

Outro aspecto importante, é a relação de compromisso observada entre QoS de detecção e o período de monitoramento. Períodos de monitoramento pequenos impli-



(a) TM média

(b) TMR médio

(c) TD médio

Figura 3.2: QoS da Detecção Adaptativa em Sistema de Controle via *Shared Ethernet*

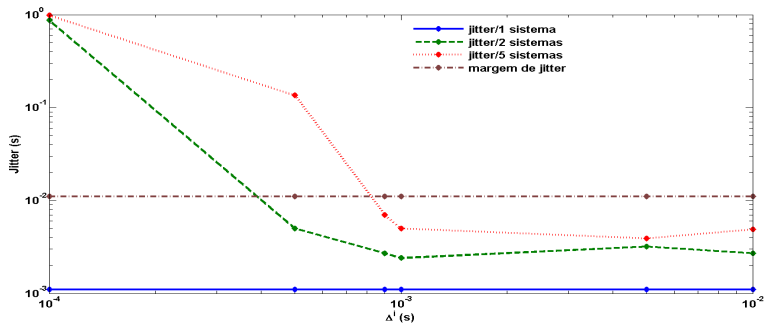
cam em menores tempo de detecção, mas, por outro lado, reduzem a confiabilidade do detector. Isto por que, a redução do período implica em uma maior disputa pelo uso da rede de comunicação, o que aumenta a magnitude e a variabilidade dos atrasos de comunicação. Isto também justifica o comportamento não linear das métricas de QoS relacionadas a confiabilidade do detector (i.e., TM e TMR).

3.1.3 Impacto da Detecção de Defeitos na Qualidade do Controle em Sistemas de Controle via Rede

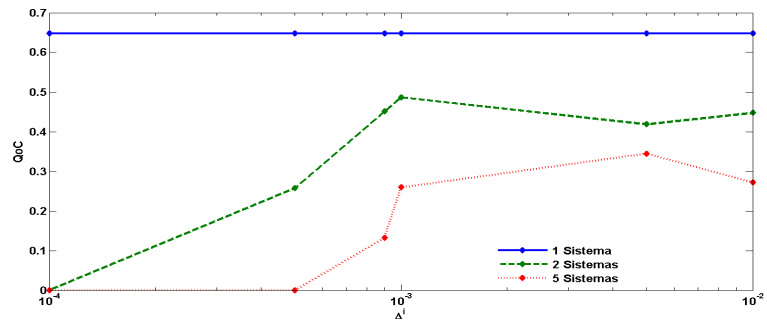
A avaliação do impacto do serviço de detecção de defeitos na malha de controle é observada em cenários nos quais a rede *Shared-Bus Ethernet* é compartilhada por 1, 2 e 5 NCS independentes e idênticos – isto é, em cenários nos quais a rede de comunicação é compartilhada por 4, 8 e 20 dispositivos, respectivamente (de modo a impor diferentes condições de carga).

Como métricas de avaliação, são usadas a margem de jitter J_m e a margem de fase aparente $\hat{\varphi}_m$ (ver Seção 1.3). A margem de *jitter* é usada como um limiar para o

atraso fim-a-fim e permite verificar o que acontece com QoC quando o estes atrasos são superiores a este limiar. A QoC, por sua vez, é obtida a partir da razão entre a margem de fase aparente e a margem de fase real, isto é: $QoC = \hat{\varphi}_m/\varphi$ (ver Seção 1.3).



(a) $Jitter$ versus Δ



(b) QoC versus Δ

Figura 3.3: QoC em função do período de monitoramento Δ

O impacto da detecção de defeitos no desempenho das malhas de controle pode ser observado nos gráficos da Figura 3.3. Para o experimento com apenas uma NCS, o *jitter* na malha de controle se mantém constante e abaixo da margem de *jitter* – ver gráfico da Figura 3.3(a). Entretanto, a QoC observada neste neste experimento é de aproximadamente 65% (ver gráfico da Figura 3.3(b)). Isto garante a estabilidade da malha, mas demonstra uma queda razoável do desempenho comparação a uma malha de controle convencional (i.e., não implementada em rede).

Para os experimentos com 2 e 5 NCS compartilhando uma mesma rede *Ethernet*, o *jitter* varia de acordo com a mudança nos períodos de monitoramento dos detectores de defeitos. Em ambos os experimentos, para períodos de monitoramento iguais ou superiores a $\Delta = 1ms$, os valores do *jitter* dos mesmos estão abaixo da margem de

jitter calculada – o que garante a estabilidade da malha de controle (ver Seção 1.3.1). Ainda assim, a QoC destas configurações são inferiores a 50% e 40% quando a rede é compartilhada por 2 e 5 sistemas, respectivamente – ver Figuras 3.3(a) e 3.3(b).

3.2 Detecção de Defeitos em Redes Veiculares

A evolução da eletrônica embarcada e das tecnologias de comunicação sem fio permitiu o desenvolvimento das redes móveis *ad hoc* (MANETs - *Mobile Ad hoc Networks*) e, posteriormente, das redes veiculares *ad hoc* (VANETs - *Vehicular Ad hoc Networks)*.

As VANETs são redes dinâmicas, não estruturadas, auto-organizáveis, com características assíncronas e distribuídas, nas quais os nós se movem em alta velocidade. Estas redes são por definição, redes de organização topológica plana, ou seja, os nós se comunicam diretamente entre si ou com outros nós através de múltiplos saltos, sem a necessidade de nenhum tipo de controle centralizado.

Apesar de conceitualmente simples, os principais problemas das VANETs estão relacionados a sua escalabilidade e a comunicação entre seus nós. Nessas redes, a comunicação é feita principalmente por difusão de mensagens *(broadcast)* e quando ocorre aumento do número de nós, o meio físico se degrada rapidamente. Além disso, o comportamento dos veículos sobre as vias e a alta velocidade com que se movem, leva a frequentes quebras nos enlaces de comunicação e consequentes mudanças na topologia da rede (NI et al., 1999; XU; GERLA, 2002).

Disputas no meio físico e frequentes mudanças na topologia da rede prejudicam o funcionamento de muitas aplicações, pois causam o aumento de atrasos e a perda de mensagens. Aplicações que dependem de comunicação fim-a-fim como, roteamento, avisos de colisão e controle semafórico, são muito sensíveis a atrasos e a perda de mensagens.

Nos últimos anos, várias propostas baseadas em organização hierárquica da rede foram apresentadas (OHTA; INOUE; KAKUDA, 2003; GUNTER; WIEGEL; GROSSMANN, 2007; FAN, 2007; SHEA, 2009). Nestas, os veículos sobre as vias são organizados em agrupamentos *(clusters)*. A principal vantagem em se utilizar este tipo de organização é que os problemas de comunicação e mobilidade dos nós são tratados dentro do agrupamento, não sendo mais propagados pela rede e, assim, do ponto de vista de cada nó e de suas aplicações, a rede se torna mais estável. Torná-la mais estável, é o primeiro passo para viabilizar o uso da comunicação em VANETs em Sistemas Inteligentes de Transporte (SITs) (MCQUEEN; MCQUEEN, 1999).

Os sistemas inteligentes de transporte baseiam-se no uso intensivo de tecnologias

de comunicação e informação em aplicações de transportes, tanto para gerenciamento como para o controle de tráfego e, atualmente, são a principal aplicação para a comunicação em VANETs (TAHA; HASAN, 2007). Grande parte destas aplicações, como gerenciamento do tráfego, avisos de colisão e direção automática, dependem de informações confiáveis em tempo real sobre o comportamento dos veículos nas vias.

No entanto, o ambiente de comunicação das VANETs é sujeito a frequentes atrasos e perda de mensagens, além de possíveis falhas em seus dispositivos. Assim, construir mecanismos que tornem os nós destas redes mais tolerantes a estas falhas é o primeiro passo no desenvolvimento de aplicações SIT mais confiáveis.

Sistemas de detecção de defeitos são um serviço fundamental para o desenvolvimento de aplicações tolerantes a falhas como, por exemplo, na comunicação de grupo, replicação de serviços e em consenso distribuído. Na última década, estes sistemas têm sido exaustivamente estudados e aplicados em redes *ad hoc* móveis (MACÊDO, 2000; MOSTEFAOUI; MOURGAYA; RAYNAL, 2003; FRIEDMAN; TCHARNY; LTD, 2005). Porém, apesar da importância em várias aplicações distribuídas em VANETs, o projeto de detectores de defeitos adequados às características de comunicação e mobilidade destas redes tem sido frequentemente negligenciada.

Nas próximas seções, descreve-se e discute-se um serviço de detecção de defeitos adaptado ao contexto de comunicação e mobilidade encontrado no ambiente das redes veiculares *ad hoc*.

3.2.1 Serviço de Detecção de Defeitos Proposto

O Serviço de detecção de defeitos proposto neste trabalho é composto por dois mecanismos: um deles permite a cada processo monitor adaptar os seus timeouts de detecção e outro que lhe permite avaliar se seus vizinhos ainda estão dentro de sua área de cobertura de comunicação. A partir do modelo de sistema adotado, estes dois mecanismos são implementados e utilizados em cada nó da rede para detectar falhas.

Modelo do Sistema. Como premissas do modelo de sistema para o serviço de detecção de defeitos proposto, assume-se que o sistema é composto por um conjunto $P = (p_1, p_2, ..., p_n)$ de processos, no qual, $n \in N$ é desconhecido, mas finito. Cada processo possui um identificador único p e representa um veículo equipado[4], que por sua vez, representa um nó na rede.

Em cada nó p há um relógio local, sincronizado pelo GPS e cuja derivação não é

[4]Um veículo é dito equipado, quando possui capacidade de comunicação, processamento embarcado, GPS *(Global Position System)* e mapa digital das vias.

significativa para as aplicações. Neles, o tempo é representado através de uma sequência T, na qual um instante de tempo t_p é um elemento de T e $t_p^{(i)}$ e indica o instante de envio do *i-ésimo heartbeat* difundido pelo processo p.

Assume-se também que os nós podem se comunicar diretamente entre si, desde que estejam sob a mesma área de cobertura de comunicação. A área de cobertura de comunicação de um nó p é representada por um círculo de raio r_p, no qual p encontra-se no centro. Uma vez que dois nós p e q estejam um sob a área de cobertura do outro, estes nós são ditos vizinhos ou membros de uma vizinhança.

A comunicação entre os nós ocorre por difusão periódica de mensagens a cada Δ segundos e uma mensagem difundida por um nó, não é necessariamente recebida por todos os seus vizinhos. Os campos desta mensagem de sinalização m são descritos a seguir:

Tabela 3.2: Campos em uma mensagem m de sinalização

Campo	Descrição
Id_p	identificador do nó p
v_p	velocidade de p em metros por segundo
$t_p^{(i)}$	instante de envio do i-ésimo heartbeat difundido pelo processo p
$p(x, y)$	posição atual do veículo p
N_p	contêm os *Ids* e *TimeStamps* dos vizinhos p

De acordo com a Tabela 3.2, cada processo p difunde periodicamente seu identificador único (Id_p), sua velocidade instantânea (v_p), seu $timestamp$ ($t_p^{(i)}$), sua posição atual ($p_{(x,y)}$) sobre o mapa digital no momento da sinalização e uma lista (N_p), que contém os Ids e $timestamps$ recebidos mais recentemente de seus vizinhos *corretos*. São considerados *corretos*, todos aqueles que não são suspeitos de terem falhado.

Assume-se também que toda difusão possui uma latência ℓ_q e um atraso $D_{(p,q)}$, que juntos indicam um tempo em segundos, necessário para o processo p enviar a mensagem para o processo q, através do canal de comunicação. A latência é calculada a partir da função descrita abaixo e utiliza o padrão IEEE 802.11s (CAMP; KNIGHTLY, 2008):

$$\ell_q = \left[O + \frac{B}{C} \right] \tag{3.1}$$

sendo que O é um valor de *sobrecarga* de tempo constante introduzida pela camada MAC *(Media Access Control)*, B indica o tamanho da mensagem em bits e C é a taxa de transmissão do canal de comunicação.

Calculada a latência, é possível obter o atraso entre a geração de uma mensagem

de sinalização e seu recebimento por um vizinho, através da equação abaixo:

$$D_{(p,q)} = t_p - (t_q^{(i)} + \ell_q) \tag{3.2}$$

onde t_p é o instante em que p recebe a sinalização, $t_q^{(i)}$ é o *timestamp* no qual q difundiu a mensagem e ℓ_q é a latência em segundos.

Modelo de Falhas. No modelo de falhas, assume-se que um processo pode apresentar dois tipos de falha: falha por parada *(crash-fault)* ou falha por abandono da via. Quando um processo é identificado como faltoso, ele é inserido em uma lista de suspeitos de falha.

Uma falha por parada indica que o nó não é mais capaz de difundir ou receber mensagens e não retornará ao sistema até que, por exemplo, seu equipamento de comunicação seja trocado. Já uma falha por abandono ocorre toda vez que um veículo abandona a via. Neste caso, o processo continua difundindo ou recebendo mensagens, porém, estas mensagens não serão consideradas válidas pelas aplicações que estão sendo executadas nos veículos que se encontram sobre as vias. Um processo que apresenta uma falha por abandono volta a ser considerado *correto*, assim que retornar a uma via.

Um processo monitor p insere outro processo monitorado q em sua lista de suspeitos, se e somente se, p não recebe dentro do *timeout* de q, uma mensagem de sinalização que indique que este processo é *correto*. Esta mensagem pode ser recebida diretamente do processo q ou, através do campo N_p, enviado por outro vizinho do processo p.

Eventualmente, o detector de defeitos pode cometer um erro e um processo *correto*, ou seja não defeituoso, pode ser inserido indevidamente na lista de suspeitos. Este processo é denominado falso suspeito e representa um erro de detecção do detector de defeitos. Uma vez que este tipo de erro pode ocorrer, assume-se que o detector não é perfeito e pode se equivocar, adicionando à lista de suspeitos processos *corretos* ou considerando processos faltosos como *corretos*.

Outro aspecto deste detector, é que nele não são consideradas nem tratadas falhas maliciosas ou bizantinas, nas quais os processos que falham, mas continuam a difundir mensagens que podem prejudicar o funcionamento do sistema. Tais falhas necessitam de outros mecanismos para detecção e tratamento, os quais não são foco desta proposta.

Adaptação do Timeout. As redes veiculares *ad hoc* estão sujeitas a quebras dos enlaces, perdas de mensagens e variações no atraso de comunicação entre os nós. Por isso, uma característica desejável aos detectores de defeitos a serem utilizados em VANETs, é que eles adaptem seus tempos de espera *timeouts* juntamente com as variações

na carga de comunicação, conforme proposto em (MACÊDO, 2000).

Detectores de defeitos tradicionais utilizam a troca periódica de sinalizações *(heartbeats)* entre os processos e, de modo geral, pressupõem que eles pertencem a uma rede completamente conectada e na qual, o atraso na comunicação é conhecido (DOLEV et al., 1997; MOSTEFAOUI; MOURGAYA; RAYNAL, 2003). Caso um processo p, não receba uma sinalização de outro processo q, dentro de um determinado *timeout*, o processo p considera q suspeito de falha, ou seja, que este parou de funcionar.

A utilização de sinalização em intervalos fixos não é adequada ao ambiente das redes móveis sem fio, pois leva a um grande número de suspeições por não considerar as variações na carga de comunicação, nem as constantes partições de enlaces nessas redes (CHEN; TOUEG; AGUILERA, 2002; TAI; TSO; SANDERS, 2004).

No serviço de detecção de defeitos proposto, um processo monitor p, adapta o tempo de espera TO_q de cada vizinho q toda vez que recebe uma sinalização deste vizinho. Para adaptar este tempo, cada processo p utiliza a seguinte função:

$$TO_q = \Delta + A_q + SM_{(p,q)}, \tag{3.3}$$

na qual Δ é o período de sinalização, A_q é a média dos atrasos das n últimas sinalizações de q e $SM_{(p,q)}$ é um tempo em segundos que varia em função da distância entre estes processos.

Para calcular a componente A_q, cada processo p mantém um histórico H_n com o atraso $D_{(p,q)}$ das últimas n sinalizações de q. O tamanho deste histórico afeta o resultado da média e, portanto, o comportamento do detector, tornando-o mais ou menos tolerante às variações na carga da rede.

$$A_q = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[D_{(p,q)}\right]^2}, \tag{3.4}$$

A componente $SM_{(p,q)}$ faz com que o valor do *timeout* aumente à medida que dois nós vizinhos se afastam um do outro e é calculada da seguinte forma:

$$SM_{(p,q)} = \begin{cases} \alpha & \text{se} \quad d(p,q) > r_p \\ \alpha + \left[\kappa \frac{d(p,q)}{r_p}\right] & \text{se} \quad d(p,q) \leq r_p, \end{cases} \tag{3.5}$$

na qual, α é uma constante que oferece uma tolerância mínima no atraso entre duas sinalizações consecutivas, $d(p,q)$ representa a distância euclidiana entre estes nós, r_p é o raio de comunicação do nó p e κ é um fator pondera a importância relativa entre

estas distâncias aos outros termos da equação.

À medida que dois vizinhos se afastam um do outro, o valor obtido em $SM_{(p,q)}$ faz com que o *timeout* entre eles aumente. Admite-se esta hipótese, pois quanto mais próximos se encontram dois vizinhos, maior a probabilidade que compartilhem a mesma vizinhança e, portanto, as mesmas condições de comunicação. No entanto, esta premissa não pode ser mantida quando eles se afastam demasiadamente – neste caso, não é possível estimar com precisão adequada a distância entre os nós e, portanto, o fator de importância relativa entre os nós κ é igual a zero. Além disso, com o aumento da distância, aumenta também a atenuação física do sinal de rádio *(fadding)*, o que colabora na degradação da comunicação entre eles, elevando a possibilidade de perda e atrasos de mensagens.

Algoritmo de Detecção de Conectividade. Antecipar, em um determinado momento, se um enlace ainda é válido pode colaborar com a aplicação de várias formas, permitindo, por exemplo, que rotas de comunicação alternativas entre origem-destino sejam encontradas antes que o enlace seja partido ou identificar falhas reais, nas quais os processos ainda estejam dentro da área de comunicação um do outro, mas não exista comunicação entre eles. O algoritmo de detecção de conectividade $DC_{(p,q)}$, descrito a seguir, é uma tarefa executada em todos os processos e tem duas funções: i) antecipar uma possível perda de conectividade entre dois processos e ii) evitar que uma quebra de enlace devido a saída de um vizinho q da área de cobertura de comunicação de p, seja confundida com uma falha.

Algorithm 3.1 Detector de Conectividade

1: $P^p_{atual} \leftarrow$ posição atual de p
2: $P^q_{ultima} \leftarrow$ última posição conhecida de q
3: $P^q_{estimada} \leftarrow$ calcular a posição estimada para o processo q em função de P^q_{ultima}
4: **if** $(|P^p_{atual} - P^q_{estimada}|) < r_p$ **then**
5: **return** true;
6: **else**
7: O processo p retira q de sua lista de vizinhos
8: **end if**
9: **return** false;

No Algoritmo 3.1, um processo monitor p verifica se um processo monitorado q ainda se encontra dentro de sua área de cobertura de comunicação. Para isso, ele calcula a posição estimada de q a partir da última posição conhecida deste processo e da posição atual em que p se encontra (linhas 2 à 5). Caso a posição estimada de q esteja fora do raio de comunicação do nó p, o vizinho q é removido da vizinhança de p (li-

nhas 6 à 9). O detector de conectividade é utilizado junto ao algoritmo de detecção de defeitos, colaborando na tarefa de diferenciar se um vizinho é faltoso ou se apenas saiu da área de cobertura de comunicação do processo monitor.

Algoritmo de Detecção de Defeitos. O serviço de detecção de defeitos proposto executa paralelamente três tarefas. Estas tarefas são descritas no Algoritmo 3.2, no qual, um processo p monitora o processo q.

Algorithm 3.2 Detector de Defeitos

```
1: task T1:
2:     loop
3:         when receber(q,m)
4:             Atualizar dados do processo q
5:             t_q^(i) ← último timestamp recebido do processo q
6:             τ_q ← timestamp mais recentemente do processo q contido no campo N_p
7:             Calcula TO_q
8:         end when
9:     end loop
10: end task
11: task T2:                                          ▷ Detecção de Defeitos
12:     loop
13:         when (t_p − t_q^(i)) > TO_q                ▷ timeout expirou
14:             if q ∉ lista de suspeitos then
15:                 Inserir q na lista de suspeitos de p
16:             end if
17:         end when
18:     end loop
19: end task
20: task T3:                                          ▷ Recuperação de Falhas
21:     loop
22:         if (DC_p(q)=True) and (q ∈ lista de suspeitos) then
23:             if (DC_p(q)=True) and ((t_p − τ_q) > TO_q) then
24:                 Retirar q da lista de suspeitos
25:             end if
26:         end if
27:     end loop
28: end task
```

Na tarefa T1, assim que um processo p recebe uma mensagem de sinalização, ele atualiza os dados deste vizinho e calcula seu novo *timeout* TO_q. Em T2, o processo monitor verifica se há entre seus vizinhos não suspeitos, algum que tenha ultrapassado o *timeout* (linhas 12 e 13), e se isto for verdadeiro, insere este vizinho na lista de suspeitos (linha 15), caso este ainda não tenha sido inserido nesta lista. Na tarefa T3

ocorre a recuperação de falsas suspeitas, retirando desta lista, aqueles processos cujo processo monitor tenha recebido uma nova mensagem de sinalização (linhas 20 a 24).

3.2.2 Avaliação do Detector de Defeitos Proposto

A avaliação do serviço de detecção de defeitos proposto foi feita utilizando o simulador de rede OMNET++ (VARGA et al., 2001), sob condições de mobilidade de veicular realísticas, obtidas do simulador de tráfego AIMSUN2000 (TRANSPORT SIMULATION SYSTEMS, 2000).

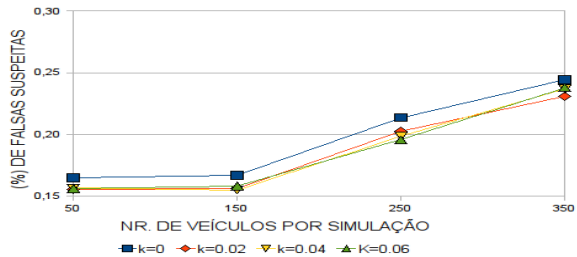
Cada simulação tem duração de 100s e assume-se um período de sinalização $\Delta = 0,1s$. O tamanho do histórico de atraso $H_n = 100$, com $\alpha = 0,02s$ e κ variando entre $0s$ e $0,06s$. Foram realizadas quatro simulações para cada valor de κ. Executar simulações com diferentes valores para esta constante, tem por objetivo verificar se esta colabora na melhoria do desempenho do detector de defeitos. Assume-se também que a sobrecarga de tempo da camada MAC é $O = 0,01s$ e o canal de comunicação possui uma taxa de transferência $C = 2Mbps$.

O cenário de simulação consiste em uma pista de mão única de $4000m$ de comprimento, na qual os veículos se movem a velocidades entre 10m/s e 22m/s. Utilizou-se em cada simulação uma densidade de tráfego diferente, são elas: 50, 150, 250 e 350 veículos sobre a pista durante todo período de simulação. Durante a simulação 20% dos nós sofrem falhas por parada *(crash-fault)*, estas falhas ocorrem aleatoriamente ao longo da simulação e um processo faltoso não retorna ao sistema.

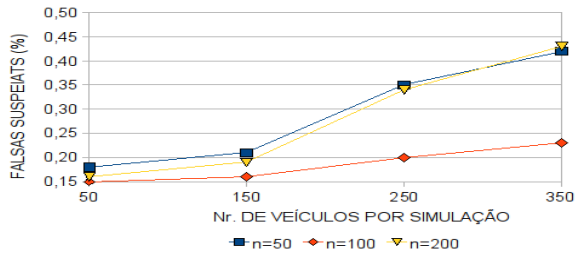
O objetivo das simulações é avaliar a eficiência do detector de defeitos proposto em relação a três métricas: i) número de falsas suspeitas, ii) tempo médio de detecção de defeitos por parada *(crash-fault)* e iii) tempo médio para recuperação de falsas suspeitas. Uma quarta métrica é utilizada para avaliar a influência do tamanho da amostra H_n na geração de falsas suspeitas. O objetivo desta métrica é mostrar a importância da correta escolha do tamanho de n no desempenho do detector de defeitos.

Simulações e Discussão dos Resultados. Um detector de defeitos é considerado mais confiável na medida que o número de suspeitas indicadas por ele é o mais próximo ao número real de falhas. Já sua eficiência está relacionada ao seu tempo de resposta, ou seja, o tempo necessário para reconhecer um defeito ou para recuperar uma falsa suspeita. Quanto menores forem estes tempos, mais eficiente é o detector.

A confiabilidade do detector pode ser avaliada através da Figura 3.4(a), na qual observa-se a relação entre o percentual de falsas suspeitas e o número de veículos na via. Note-se que, enquanto a densidade da rede aumentou sete vezes, o número de



(a) *Falsas Suspeitas X Densidade do Tráfego*



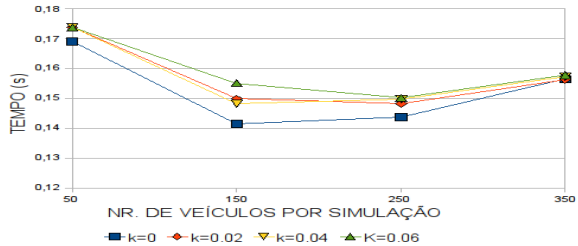
(b) *Tamanho do histórico H_n X Falsas Suspeitos*

Figura 3.4: Avaliação da Confiabilidade do Detector de Defeitos

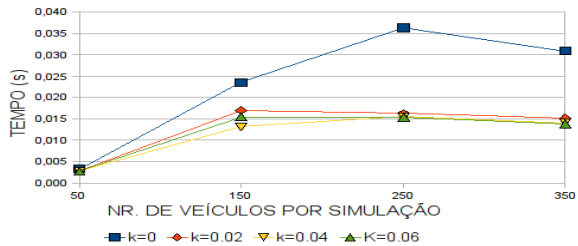
falsas suspeitas aumentou apenas cinquenta por cento. Isto demonstra a estabilidade, escalabilidade e confiabilidade do detector de defeitos diante da variação da carga de comunicação da rede e das mudanças na densidade de tráfego, que passou de esparsa para densa.

Nota-se na Figura 3.4(b) que a quantidade de amostras de atrasos entre as sinalizações, armazenadas em H_n, afeta o desempenho do detector. Os critérios para determinar a quantidade das amostras dependem de fatores como o período de sinalização e as restrições temporais das aplicações que utilizam o detector. Observa-se que amostras muito pequenas deixam o detector de defeitos muito sensível às mudanças de carga de comunicação na rede. Por outro lado, amostras muito grandes, amenizam demasiadamente a ação do detector. Em ambos os casos, a confiabilidade do detector diminui. Assim, a escolha de valores de n deve ser criteriosa e depende de estudos de casos.

Note-se na Figura 3.5(a), na qual, avalia-se o tempo médio de detecção de defeitos, que o aumento da densidade reduz o tempo de detecção. Isto porque se, por um lado, o aumento da densidade degrada a comunicação na rede, elevando a perda de mensagens e os atrasos, por outro, um processo monitor p aumenta sua chance de receber informações de um vizinho q através de outros processos monitorados.



(a) *Tempo Médio de Detecção de Falha*



(b) *Tempo Recuperação de Falsas Suspeitas*

Figura 3.5: Tempo de Resposta do Detector de Defeitos

Receber informações de um processo monitorado q através de mensagens enviadas por outros vizinhos, colabora na redução do tempo de detecção de defeitos e na redução do tempo médio de recuperação de falsas suspeitas, como mostra a Figura 3.5(b) – ver Cambruzzi e outros (2016) para mais detalhes. Observa-se nessa figura que o tempo médio de recuperação de falhas aumenta até encontrar um ponto no qual se estabiliza e começa a diminuir, mesmo com o aumento contínuo da densidade de veículos na via. Novamente, isto se deve ao fato de que juntamente com o aumento da densidade dos nós, aumenta a chance dos processos monitores receberem mais indicações sobre o estado dos processos monitorados através de seus vizinhos. Outro dado que se pode observar nessa figura é que, o tempo de recuperação de falsas suspeitas é muito baixo em relação do período de sinalização, demonstrando mais uma vez a eficiência do detector diante de possíveis erros de detecção.

Por fim, outro aspecto importante a ser observado nas Figuras 3.4 e 3.5 é a influência da variação da constante κ no desempenho do detector. O aumento do valor da constante κ não afeta significativamente o tempo médio de detecção, mas colabora na redução do percentual de falsas suspeitas (ver Figura 3.4(a)) e também contribui para a redução do tempo de recuperação destas falsas suspeitas (ver Figura 3.5(b)).

3.2.3 Considerações Finais

A capacidade de adaptar o detector de defeitos a carga de comunicação da rede é uma condição importante para várias aplicações, principalmente aquelas que possuem restrições temporais críticas. Os sistemas inteligentes de transporte possuem várias destas aplicações como, avisos de colisão, controle de passagem em cruzamentos cegos, veículos automáticos, entre outros. Estas aplicações dependem de informações confiáveis e em tempo real sobre as condições do tráfego no entorno de cada veículo e devem ser robustas o suficiente para tolerar falhas, atrasos e perdas de mensagens.

O serviço de detecção de defeitos proposto neste trabalho busca oferecer às aplicações informações mais confiáveis sobre o estado da rede. Para isso, ele procura se adequar às frequentes mudanças das condições de comunicação e mobilidade encontradas nas VANETs. Os resultados obtidos através das simulações demonstram que o detector possui boa capacidade de adaptação ao ambiente das VANETs, detectando falhas rapidamente e, ao mesmo tempo, apresenta um baixo número de falsas suspeitas, cuja recuperação também é rápida. Este sistema também se mostra escalável, mantendo sua estabilidade e desempenho diante das variações da densidade do tráfego. Esta capacidade de manter-se estável, mesmo diante de mudanças na densidade do tráfego, é uma característica fundamental para serviços de detecção de defeitos projetados para o ambiente das VANETs. Nessas redes, o tráfego muda constantemente, seja ao longo do dia, seja ao longo de uma mesma via, afetando de modo significativo a comunicação entre seus nós.

Apesar da importância de desenvolver serviços de detecção de defeitos mais específicos para o ambiente das VANETs, são poucas as propostas existentes. A solução mostrada e discutida nas seções anteriores contribui neste sentido e oferece um conjunto de mecanismos para que aplicações que dependem de comunicação em VANETs possam utilizá-la de modo mais confiável e seguro.

3.3 Detecção Auto-gerenciável de Defeitos

Detectar defeitos por parada (i.e. *crash*) de componentes é uma questão básica para o funcionamento de muitos protocolos e algoritmos usados na construção de sistemas distribuídos confiáveis. Por exemplo, em um esquema de replicação passiva, o defeito de uma réplica primária precisa ser prontamente detectado para que uma das réplicas secundárias assuma o papel da réplica faltosa com o mínimo de impacto para as aplicações ou serviços distribuídos.

A detecção de defeitos por *crash* geralmente considera que nós monitorados en-

viam periodicamente mensagens, ditas *heartbeats*, as quais indicam seu estado para nós monitores (ver Seção 2.2). Um nó monitor determina um intervalo de tempo (i.e. *timeout* de detecção) durante o qual esperará pela chegada da mensagem de *heartbeat*. Se a mensagem de *heartbeat* não chega dentro do *timeout* de detecção, o nó monitor acreditará que o nó monitorado falhou. Esse modelo de detecção de defeitos depende das restrições temporais relacionadas ao processamento e transmissão das mensagens de monitoramentos trocadas entre os módulos do detector de defeitos.

Em um sistema distribuído assíncrono, os limites temporais para processamento e transmissão das mensagens são desconhecidos (LAMPORT; LYNCH, 1989), o que torna impossível solucionar certos problemas de tolerância a falhas de forma determinística (FISCHER; LYNCH; PATERSON, 1985) – restrições estas que são herdadas por aplicações distribuídas baseadas em tecnologias modernas para automação, como Internet das Coisas[5] e Sistemas Cibernéticos-Físicos[6], por exemplo.

Para responder a essa impossibilidade, Chandra e Toueg (1996) introduziram o conceito de detectores de defeitos não confiáveis – os quais podem apontar como defeituosos, nós corretos, e, por outro lado, deixar de apontar nós que efetivamente falharam. Chandra e Toueg (1996) demonstraram como, encapsulando certo nível de sincronia, detectores de defeitos não confiáveis podem ser usados para solucionar problemas fundamentais em sistemas distribuídos assíncronos – e.g. consenso distribuído (GUERRAOUI et al., 2000) e difusão atômica (DÉFAGO; SCHIPER; URBÁN, 2004). Apesar da grande importância do trabalho de Chandra e Toueg (1996) para o entendimento e solução de problemas fundamentais em sistemas distribuídos, a ausência de limites temporais dos modelos assíncronos impõe grandes desafios práticos para a implementação de detectores de defeitos.

Um desses desafios é decidir valores apropriados para o *timeout* de detecção. *Timeouts* muito longos tornam a detecção dos defeitos lenta e compromete a resposta do sistema durante a ocorrência de falhas. Por outro lado, *timeouts* muito curtos podem degradar a confiabilidade do detector de defeitos e prejudicar o desempenho do sistema, uma vez que muitos algoritmos e protocolos, que utilizam a informação do detector de defeitos, podem realizar processamento e troca de mensagens adicionais por conta de falsas suspeitas de falhas. Nesse contexto, Macêdo (2000) propõe um mecanismo denominado CTI (*Connectivity Time Indicator*), o qual é inserido em uma abordagem de detecção de defeitos não confiável, com o intuito de sugerir *timeouts* de detecção dinâmicos, os quais variam de acordo com as condições de carga do ambiente

[5] do Inglês *Internet of Things* (IoT) – ver, por exemplo, Atzori, Iera e Morabito (2010) e Whitmore, Agarwal e Xu (2015) para maiores detalhes

[6] *Cyber-Physical Systems* (CPS), ver Kim e Kumar (2012) para maiores detalhes

distribuído. Esse trabalho é seguido por muitos outros com o intuito de embutir o uso de estimadores de *timeouts* na implementação de detectores de defeitos – como por exemplo, Bertier, Marin e Sens (2002), Macêdo e Lima (2004), Nunes e Jansch-Pôrto (2004), Falai e Bondavalli (2005), entre outros. O papel desses estimadores é sugerir, em tempo de execução, valores adequados para o *timeout* de detecção, de modo a tornar a detecção dos defeitos mais rápida com o mínimo possível de impacto para a confiabilidade do detector de defeitos. No entanto, tais trabalhos não consideram o ajuste dinâmico do período de monitoramento, outro fator importante para o desempenho dos detectores de defeitos – principalmente quando os custos computacionais da detecção precisam ser considerados.

Outro problema é que a especificação dos detectores de defeitos de Chandra e Toueg (1996) não foca em aspectos pertinentes a qualidade do serviço da detecção de defeitos, definindo propriedades difíceis de serem avaliadas na prática, por exemplo: "*em algum momento as falhas dos processos serão detectadas por algum processo correto*"[7]. Por conta disto, Chen, Toueg e Aguilera (2002) definiram métricas de qualidade de serviço para detecção de defeitos, as quais têm sido usadas para avaliar a velocidade e a precisão de diferentes implementações de detectores de defeitos. Com isso, o trabalho do projetista é definir um período de monitoramento e usar um estimador de *timeout* que consiga entregar um serviço de detecção de defeitos com um nível de qualidade de serviço adequado aos requisitos das aplicações.

Outro aspecto de projeto importante na concepção de detectores de defeitos, é compatibilizar o custo do serviço de detecção com as características dinâmicas dos ambientes computacionais modernos e de suas aplicações. Em ambientes distribuídos modernos, sujeitos a condições de carga variadas, a mudanças dinâmicas de requisitos de qualidade, ou a variações na disponibilidade de recursos, a configuração dos parâmetros operacionais dos detectores de defeitos, considerando métricas de qualidade de serviço e custo computacional da detecção, é uma atividade difícil de ser realizada. Para garantir, por exemplo, uma recuperação rápida na presença de componentes defeituosos, o período de monitoramento deve ser tão curto quanto possível. Todavia, períodos de monitoramento muito curtos podem incrementar demasiadamente o consumo de recursos computacionais, comprometendo o tempo de resposta das aplicações e diminuindo a eficiência e a velocidade dos mecanismos de detecção de defeitos e de recuperação.

Nesse contexto, Chen, Toueg e Aguilera (2002) propõem um procedimento para a configuração *off-line* dos detectores de defeitos. Os mesmos sugerem que tal pro-

[7] tradução do inglês, *"eventually every process that crashes is permanently suspected by some correct process"* (CHANDRA; TOUEG, 1996, p. 232).

cedimento pode ser re-executado durante o funcionamento do detector, quando as características de carga do ambiente computacional mudam. Todavia, os efeitos dessa re-execução no desempenho dos detectores de defeitos não foram avaliados.

Bertier, Marin e Sens (2003) comentam brevemente um procedimento baseado em consenso para ajustar dinamicamente o período de monitoramento quando certas condições de carga são verificadas. Entretanto, não detalham a solução, nem avaliam a mesma considerando métricas de QoS de detecção.

Mills e outros (2004), Xiong e outros (2006) e So e Sirer (2007) exploram a configuração dinâmica de detectores de defeitos. Entretanto, consideram que o comportamento do ambiente computacional não muda e não demonstram como dinamicamente configurar os detectores de defeitos usando métricas de qualidade de serviço, como tempo de detecção, duração da falsa suspeita e intervalo entre falsas suspeitas.

Dixit e Casimiro (2010) propuseram uma abordagem de detecção de defeitos que usa métricas de qualidade de serviço para o ajuste do *timeout* de detecção. Entretanto, os mesmos não consideram o ajuste dinâmico do período de monitoramento, outro aspecto importante para a adequação do custo computacional relacionado ao serviço de detecção de defeitos.

Diferentes dos trabalhos apresentados na literatura, os detectores de defeitos propostos por Sá e Macêdo (2010a, 2010b) são os primeiros capazes de auto-configurar seus parâmetros operacionais em tempo de execução, em resposta às mudanças no ambiente computacional ou nas aplicações, observando, para tanto, os requisitos de qualidade de serviço definidos pelo usuário. Sistemas com tais características são ditos autonômicos e a auto-configuração é uma de suas propriedades básicas (HUEBSCHER; MCCANN, 2008).

A maior dificuldade na implementação de detectores auto-configuráveis de defeitos é a modelagem da dinâmica do sistema distribuído – a qual é difícil de caracterizar usando funções de distribuição de probabilidades quando, por exemplo, ambientes com condições de carga variadas são considerados. Para modelar tal comportamento dinâmico dos sistemas distribuídos, este trabalho utiliza a teoria de controle realimentado, comumente aplicada na área de automação de sistemas industriais (OGATA, 1995).

Assim, o restante desta Seção: (a) discute o modelo de sistema adotado na concepção da proposta de detecção autonômica de defeitos (Seção 3.3.1); (b) apresenta os detalhes de implementação da detecção autonômica (Seção 3.3.2); e (c) descreve os experimentos realizados para verificação do desempenho do detector autonômico, quando comparado com abordagens adaptativas existentes na literatura (Seção 3.3.3).

3.3.1 Modelo de sistema

Para a implementação do serviço de detecção autonômica de defeitos, considera-se um modelo de sistema distribuído constituído por um conjunto finito de n nós $\Pi = \{p_1, p_2, ..., p_n\}$. Esses nós são interconectados através de canais de comunicação não confiáveis, os quais podem duplicar ou perder mensagens. Se uma mensagem enviada por um nó do sistema é corrompida, a mesma será descartada. Além disso, se um nó p_i envia, para um nó correto p_j, uma mesma mensagem sucessivas vezes, em algum momento tal mensagem será recebida com sucesso por p_j – i.e., os nós interagem através de canais de comunicação do tipo *fair-lossy* (LYNCH, 1996, p. 691–732).

Os nós possuem acesso aos seus relógios locais, não são assumidos relógios sincronizados e as taxas de desvio desses relógios em relação ao tempo real são valores muitíssimo menores que os tempos de transmissão e processamento das mensagens – sendo desprezados nas estimativas dos atrasos. Além disso, não são assumidos limites temporais para processamento e transmissão das mensagens.

Os nós do sistema podem falhar por *crash* e o modelo de sistema não considera falhas bizantinas (LAMPORT; SHOSTAK; PEASE, 1982). Cada nó tem acesso a um módulo local de um serviço de detecção de defeitos, o qual provê informações, possivelmente não confiáveis, sobre o estado dos demais nós do sistema. Este serviço de detecção de defeitos é concebido considerando o estilo de monitoramento *pull* (ver Seção 2.2.1). As mensagens de monitoramento trocadas entre os módulos do serviço detecção de defeitos são assinaladas com números sequenciais.

Os recursos do ambiente computacional e os padrões de carga de suas aplicações podem mudar dinamicamente. Além disso, o serviço de detecção de defeitos não possui qualquer conhecimento sobre as características do ambiente computacional. Para o serviço de detecção, as únicas informações disponíveis são aquelas obtidas através das mensagens de monitoramento trocadas entre os seus módulos.

3.3.2 A proposta de detecção autonômica de defeitos

A proposta de detecção autonômica de defeitos considera a implementação de um gestor autonômico (ou controlador), o qual observa o comportamento do serviço básico de detecção de defeitos (elemento gerenciado ou planta). Então, baseado nos requisitos de qualidade de serviço de detecção (i.e. TD_U, TM_U e TMR_L) e nas restrições de consumo de recursos (RC_D) previamente definidos, esse gestor autonômico calcula o período de monitoramento (Δ) e o *timeout* de detecção (TO), os quais são usados por um nó monitor p_i para checar o estado de um nó monitorado p_j (ver Figura 3.6).

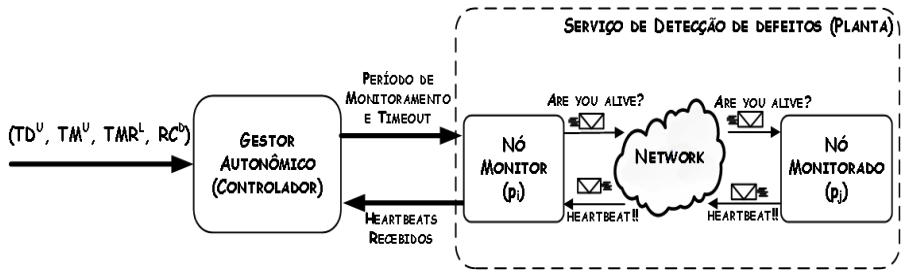


Figura 3.6: Abordagem de detecção autonômica de defeitos

O gestor autonômico executa três tarefas básicas: (i) sensoriamento de características do ambiente computacional e do desempenho do serviço de detecção – para observar o atendimento das metas de desempenho definidas pelos usuários ou aplicações; (ii) regulação do *timeout* de detecção – para atender a relação de compromisso entre a precisão e a velocidade da detecções de defeitos; (iii) regulação do período de monitoramento – para atender a relação de compromisso entre a velocidade e o custo computacional das detecções. Uma discussão mais detalhada sobre a implementação de cada uma destas tarefas é apresentada a seguir.

Sensoriamento do Ambiente Computacional e do Serviço de Detecção de Defeitos. A tarefa de sensoriamento consiste em estimar os atrasos no ambiente computacional e o desempenho do detector em termos de qualidade de serviço de detecção e de suposições a respeito do consumo de recursos. Para tanto, a tarefa de sensoriamento extrai essas informações a partir do comportamento das mensagens de monitoramento trocadas entre os módulos do serviço de detecção de defeitos. As estratégias e conceitos definidos e utilizados no sensoriamento são descritos a seguir.

Sensoriamento do Ambiente Computacional. O sensoriamento do consumo dos recursos do ambiente computacional é o ponto de partida para que o gestor autonômico realize a regulação dos parâmetros do detector de defeitos. Tal atividade é importante para a regulação dos períodos de monitoramentos usados pelos módulos do serviço de detecção.

Quando não existem informações disponíveis a respeito das características do ambiente computacional, os atrasos fim-a-fim, observados na transmissão das mensagens de monitoramento do detector de defeitos, podem dar um indicativo do suposto consumo de recursos no ambiente computacional. Para tanto, a cada *heartbeat* recebido, é possível mensurar o atraso de ida-e-volta (rtt) de uma mensagem de monitoramento por $rtt^{(k)} = r^{(k)} - s^{(k)}$, em que $s^{(k)}$ e $r^{(k)}$ são, respectivamente, os instantes de envio

de um *"are you alive?"* $aya^{(k)}$ e de recebimento de seu respectivo *heartbeat* $hb^{(k)}$ – ver Figura 3.7.

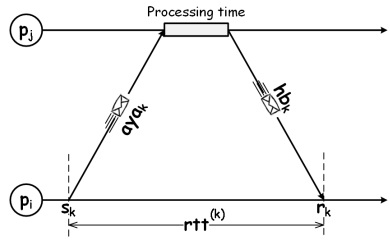
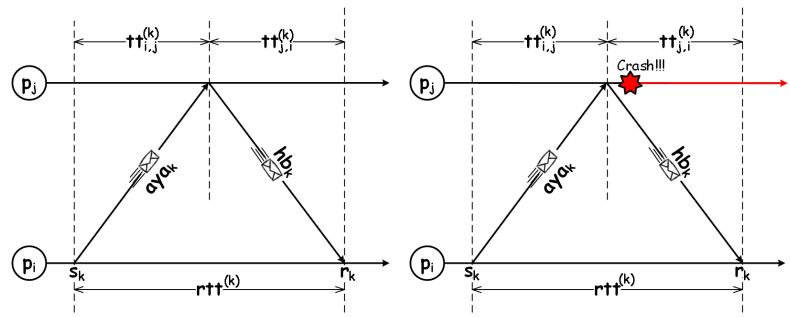


Figura 3.7: Estimativa do atraso de ida-e-volta rtt

Perdas de mensagens dificultam a interação entre os nós do sistema e impactam diretamente na percepção de um nó monitor p_i, a respeito do estado de um nó monitorado p_j. Por exemplo, p_i pode suspeitar da falha de p_j por causa da perda de uma mensagem de monitoramento.

No caso da detecção de defeitos, mensurar os atrasos de ida-e-volta a partir das diferenças entre os instantes de recebimento e de envio pode ser uma tarefa onerosa ou prejudicar a reação do gestor autonômico, quando perdas de mensagens precisam ser consideradas. Na prática, é possível implementar uma estratégia de retransmissão na qual o atraso de ida-e-volta é mensurado considerando o instante de envio do primeiro aya até o instante de tempo em que um hb é finalmente recebido. Entretanto, esse tipo de implementação tem implicações no custo de monitoramento. Por exemplo, retransmissões desnecessárias podem ocorrer se o nó monitor não sabe se o nó monitorado falhou ou se a mensagem foi perdida – o que é um caso comum em muitos sistemas distribuídos típicos. Além disso, o custo com retransmissões pode ser evitado, uma vez que, o monitoramento de defeitos é periódico, o que significa que a perda de uma mensagem de monitoramento em um ciclo de monitoramento pode ser compensada pelo envio de uma (nova) mensagem em um ciclo de monitoramento posterior. Outra alternativa seria obter uma estimativa da probabilidade de perda de mensagens. Entretanto, esta abordagem pode levar o gestor autonômico a atuar de forma ineficiente, uma vez que, se o ambiente possui características dinâmicas, isto é as distribuições de probabilidade dos atrasos computacionais não são conhecidas a priori e podem mudar dinamicamente. Assim, um ponto importante na construção do gestor autonômico é obter uma medida eficiente e que evite custos desnecessários para o monitoramento das falhas. Para tanto, é necessário analisar, do ponto de vista do serviço de detecção, o atraso na interação entre pares de nós do sistema. O atraso de interação é um conceito introduzido em (SÁ; MACÊDO, 2010a) e é apresentado a seguir.

Em sistemas distribuídos, quando um nó monitor p_i recebe um *heartbeat* de um nó monitorado p_j, o mesmo não tem qualquer garantia de que p_j ainda esteja funcionando. Isto porque o recebimento de um *heartbeat* carrega apenas informações sobre o estado de p_j no passado. Portanto, se p_i recebe um *heartbeat* em um instante $^{(k)}$, este sabe apenas que p_j esteve funcionando corretamente até o instante $r^{(k)} - tt_{j,i}^{(k)}$, em que $tt_{j,i}^{(k)}$ é o tempo de viagem de $hb^{(k)}$ de p_j para p_i (ver Figura 3.8).



(a) p_j ainda está correto em r_k (b) p_j está falho em r_k

Figura 3.8: Interação entre p_i e p_j

Um fato importante, na análise da interação entre os processos p_i e p_j, é que quanto maior o intervalo de tempo decorrido desde o recebimento do último *heartbeat* maior é a incerteza de p_i a respeito do estado de p_j. Com isso, a partir desta discussão, é introduzido o conceito de *intervalo de incerteza*:

Definição 3.1 ***Intervalo de incerteza*** *(uti): é uma medida do envelhecimento da informação obtida por um nó monitor p_i a respeito do estado de um nó monitorado p_j, podendo ser calculada, em dado instante t, por:*

$$uti(t) = t - \left[r^{(u)} - tt_{j,i}^{(u)} \right]$$

em que $r^{(u)}$ e $tt_{j,i}^{(u)}$ são, respectivamente, o instante de chegada e tempo de viagem do último heartbeat recebido $hb^{(u)}$ até o instante t.

A partir da Definição 3.1, obtem-se a primeira propriedade a respeito do intervalo de incerteza:

Propriedade 3.1 ***Influência do atraso fim-a-fim em*** *uti: A magnitude do intervalo de incerteza, estimada por um nó monitor p_i, depende diretamente da magnitude do atraso fim-a-fim (tt) entre um nó monitorado p_j e o nó p_i.*

O gestor autonômico estima uti nos instantes de envio dos "*are you alive?*". Assim, a cada intervalo k, o mesmo calcula $uti^{(k)} = uti(s^{(k)})$, ou:

$$uti^{(k)} = s^{(k)} - \left[r^{(u)} - tt_{j,i}^{(u)}\right] \tag{3.6}$$

O conceito de intervalo de incerteza possui mais três propriedades importantes:

Propriedade 3.2 ***Influência do período de monitoramento em*** uti*:* *A magnitude do intervalo de incerteza, estimada por um nó monitor p_i, depende diretamente da magnitude do período de monitoramento (δ) usado por p_i para verificar o estado de um nó monitorado p_j.*

Demonstração. Suponha que $hb^{(u)}$ foi recebido por p_i. Uma vez que $rtt^{(u)} = r^{(u)} - s^{(u)}$ e $rtt^{(u)} = tt_{i,j}^{(u)} + tt_{j,i}^{(u)}$, então, $r^{(u)} - tt_{j,i}^{(u)} = s^{(u)} + tt_{i,j}^{(u)}$. Assim, no instante $s^{(u+1)}$, o gestor autonômico estimará $uti^{(u+1)} = s^{(u+1)} - [s^{(u)} + tt_{i,j}^{(u)}]$. Dado que $\Delta^{(u)} = s^{(u+1)} - s^{(u)}$, então $uti^{(u+1)} = \Delta^{(u)} - tt_{i,j}^{(u)}$. ∎

Propriedade 3.3 ***Influência de perdas de mensagens de monitoramento em*** uti*:* *A magnitude do intervalo de incerteza, estimada por um nó monitor p_i, cresce na medida em que mensagens de monitoramento são perdidas.*

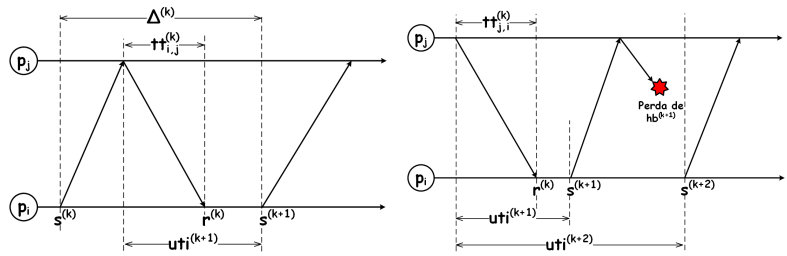
Demonstração. Suponha que após $hb^{(u)}$ ter sido recebido por um nó monitor p_i, w mensagens de monitoramento consecutivas foram perdidas. Então, usando a Equação 3.6, $uti^{(u+1)} = s^{(u+1)} - [r^{(u)} - tt_{j,i}^{(u)}]$, da mesma forma, $uti^{(u+2)} = s^{(u+2)} - [r^{(u)} - tt_{j,i}^{(u)}]$, ..., $uti^{(u+w)} = s^{(u+w)} - [r^{(u)} - tt_{j,i}^{(u)}]$. Dado que $s^{(u)} < s^{(u+1)} < ... < s^{(u+w)}$, então $uti^{(u+1)} < uti^{(u+2)} < \cdots < uti^{(u+w)}$. ∎

A Figura 3.9 ilustra uti para (a) o caso no qual mensagens não são perdidas e (b) o caso no qual *heartbeats* são perdidos.

Propriedade 3.4 ***Influência da falha de um nó monitorado*** p_j ***em*** uti*:* *A magnitude do intervalo de incerteza, estimada por um nó monitor p_i, cresce indefinidamente quando um nó monitorado p_j falha.*

Demonstração. Suponha que $hb^{(u)}$ é o último *heartbeat* recebido por um nó monitor p_i após a falha de um nó monitorado p_j. Deste modo, p_i não receberá, de p_j, qualquer *heartbeat* $hb^{(x)}$, com $x > u$. Sendo assim, na medida em que $x \to \infty$, então $s^{(x)} \to \infty$ e, consequentemente, $uti^{(x)} \to \infty$. ∎

O intervalo de incerteza permite que o gestor autonômico possa observar o impacto do atraso e da perda de mensagens sem um custo adicional para o monitoramento –



(a) uti sem perda de mensagens (b) uti com perda de mensagens

Figura 3.9: Exemplo do intervalo de incerteza uti

ver propriedades 3.1 e 3.3. Entretanto, para que este conceito possa ser utilizado, é necessário que se obtenha uma estimativa do atraso fim-a-fim (tt). Para tanto, em um dado intervalo k, estima-se $tt_{i,j}^{(k)} = tt_{j,i}^{(k)} = rtt^{(k)}/2$.

Apesar de, na prática, as mensagens de *"are you alive?"* e de *"heartbeat"* poderem ter tempos de viagem diferentes, assume-se, por questões de simplicidade, $rtt/2$ como uma estimativa para o tempo de viagem de uma mensagem de monitoramento. Observe que, para o caso da estimativa do suposto percentual consumo de recursos, um aumento ou diminuição de rtt será refletido em $rtt/2$. Dessa forma, o fato de as mensagens aya e hb poderem ter tempos de viagem diferentes tem pouca importância no contexto da gestão autonômica do serviço de detecção de defeitos.

Um inconveniente no uso de uti, na atividade de regulação do período de monitoramento, é o fato de, em cenários sem perda de mensagens ou falhas, o mesmo ser influenciado pelo período de monitoramento determinado pelo gestor autonômico, ver Propriedade 3.2. Isto porque, nesses cenários, a informação fornecida pela tarefa de sensoriamento para a tarefa de regulação de período estará *contaminada* com o valor do período de monitoramento – necessitando, assim, que tal tarefa de regulação realize um tratamento antes que a informação possa ser efetivamente usada. Para solucionar tal inconveniente, define-se o conceito de atraso de interação:

Definição 3.2 ***Atraso de interação****: é uma medida do atraso, imposto pelo ambiente computacional, para que o estado de um nó monitorado p_j seja obtido por um nó monitor p_i, podendo ser calculado em um dado ciclo de monitoramento k por:* $D^{(k)} = \left|uti^{(k)} - \Delta^{(k-1)}\right|$.

Na ausência de falhas, o atraso de interação apresenta uma propriedade importante:

Propriedade 3.5 ***Atraso de interação em ciclos de monitoramento livres de falhas****:*

Quando não existem perdas de mensagens ou falha do nó monitorado, o atraso de interação, observado por um nó monitor p_i, representa uma estimativa do atraso fim-a-fim – i.e. $D^{(k)} = rtt^{(k-1)}/2$.

Demonstração. Sendo $D^{(k)} = \left|uti^{(k)} - \Delta^{(k-1)}\right|$ e $uti^{(u)} = \Delta^{(k-1)} - tt_{i,j}^{(k-1)}$, então $D^{(k)} = \left|-tt_{i,j}^{(k-1)}\right|$. Uma vez que, $tt_{i,j}^{(k-1)} = rtt^{(k-1)}/2$ e $rtt \geq 0$, então $D^{(k)} = rtt^{(k-1)}/2$. ∎

Em cenários com perdas de mensagens de monitoramento ou falha do nó monitorado, o atraso de interação herda, do intervalo de incerteza, duas propriedades importantes:

Propriedade 3.6 ***Influência de perdas de mensagens de monitoramento no atraso de interação****: A magnitude do atraso de interação, estimada por um nó monitor p_i, cresce na medida em que mensagens de monitoramento são perdidas na interação com um nó monitorado p_j.*

Demonstração. Segundo a Propriedade 3.3, se após o recebimento de um *heartbeat* $hb^{(u)}$, uma seqüência de w mensagens de monitoramento é perdida, tem-se que $uti^{(u+1)} < uti^{(u+2)} < ... < uti^{(u+w)}$. Se $D^{(x)} = |uti^{(x)} - \Delta^{(x-1)}|$, então $D^{(u+1)} < D^{(u+2)} < ... < D^{(u+w)}$. ∎

Propriedade 3.7 ***Influência da falha de um nó monitorado p_j no atraso de interação****: A magnitude do atraso de interação, estimada por um nó monitor p_i, cresce indefinidamente quando um nó monitorado p_j falha.*

Demonstração. A partir da Propriedade 3.4, se na falha p_j, $uti \to \infty$, na medida em que $s \to \infty$, então por definição $D \to \infty$. ∎

A Propriedade 3.5 permite ao gestor autonômico usar o atraso fim-a-fim como uma suposta estimativa para o consumo de recursos. Observe que o aumento dos atrasos fim-a-fim associados às mensagens de monitoramento não necessariamente representa um aumento do suposto uso de recursos no ambiente computacional. Isto pode ser causado, por exemplo, por defeitos nos componentes do sistema (e.g. interferência nos canais, falhas de gestão de memória etc.). Entretanto, seja por conta do uso efetivo dos recursos, seja por conta de defeitos em componentes do sistema, um aumento nos atrasos fim-a-fim representa uma suposta diminuição do percentual de recursos disponíveis.

Através da Propriedade 3.6, o gestor autonômico, durante a ocorrência de perda de mensagens, percebe atrasos maiores e consequentemente estima que existem menos

recursos disponíveis. Com isso, o mesmo pode fornecer períodos de monitoramento maiores nestes casos – o que representa um aspecto importante, uma vez que, a ocorrência de perda de mensagens pode ser um indicativo de possíveis congestionamentos ou altas demandas por recursos computacionais.

A Propriedade 3.7 permite que o gestor autonômico leve o período de monitoramento para o máximo possível, após a ocorrência de falhas do nó monitorado. Este é um aspecto relevante para assegurar que o custo do serviço de detecção autonômica é reduzido quando nós monitorados falham.

Observe que o atraso de interação, além de possuir propriedades favoráveis a gestão autonômica do detector de defeitos, representa uma métrica operacional, não demandando, portanto, o uso de qualquer função de distribuição de probabilidade ou qualquer outra informação definida a priori – um aspecto importante para a regulação do período de monitoramento quando se considera ambientes com características dinâmicas.

A partir das definições acima, é importante descrever que informações são extraídas do atraso de interação para que o mesmo possa realizar uma estimativa do percentual de uso (ou de disponibilidade) de recursos no ambiente computacional. Para tanto, o gestor autonômico caracteriza o ambiente computacional através de D, de seus valores máximos e mínimos, ditos D_U e D_L, e de sua máxima variação, dita J_U. Os valores de D_L, D_U e J_U podem ser estimados assumindo o menor e o maior valor de D e sua maior variação durante a execução do detector, respectivamente.

Entretanto, se as características do ambiente variam, é possível que os valores de D_L, D_U e J_U observados pelo gestor autonômico também mudem. Com isso, considera-se um fator de esquecimento $f \in [0, 1]$ na estimativa de tais valores (f é descrito mais adiante). Assim, o gestor autonômico estima os valores de D_L, D_U e J_U usando o procedimento descrito no Algoritmo 3.3. Nesse Algoritmo, antes de estimar os atrasos, o gestor autonômico assume $D_L^{(0)} = D_U^{(0)} = rtt^{(0)}/2$ e $J_U^{(0)} = 0$ (Linhas 1–3). Em seguida, antes do envio de cada *heartbeat* $hb^{(k)}$, o gestor autonômico calcula $D_L^{(k)}$, $D_U^{(k)}$ e $J_U^{(k)}$ (Linhas 7–18). Contudo, para realizar tais estimativas, primeiramente o gestor autonômico calcula os valores do fator de esquecimento f e do atraso de interação D (Linhas 5–6). Definir f arbitrariamente pode levar o gestor autonômico a realizar ajustes inadequados em Δ, por conta da velocidade da variação de D_L, D_U e J_U. Assim, f é definido a partir do tempo máximo de detecção (TD_U) e do atraso de interação mínimo (D_L):

$$f^{(k)} = \frac{max[0, (TD_U - D_L^{(k)})]}{TD_U} \tag{3.7}$$

em que $f^{(0)} = 1$ e TD_U é definido pelo usuário (ou pelas aplicações).

Algorithm 3.3 Sensoriamento do ambiente computacional

1: $D_L^{(0)} \leftarrow \frac{rtt^{(0)}}{2}$
2: $D_U^{(0)} \leftarrow \frac{rtt^{(0)}}{2}$
3: $J_U^{(0)} \leftarrow 0$
4: **before event** $hb^{(k)}$ sending:
5: use the Equation 3.7 to compute $f^{(k)}$
6: use the Definition 3.2 to compute $D^{(k)}$
7: **if** $D_L^{(k-1)} > D^{(k)}$ **then**
8: $D_L^{(k)} \leftarrow D^{(k)}$
9: **else**
10: $D_L^{(k)} \leftarrow f^{(k)} * D_L^{(k-1)} + (1 - f^{(k)}) * D^{(k)}$
11: **end if**
12: **if** $D_U^{(k-1)} < D^{(k)}$ **then**
13: $D_U^{(k)} \leftarrow D^{(k)}$
14: **else**
15: $D_U^{(k)} \leftarrow f^{(k)} * D_U^{(k-1)} + (1 - f^{(k)}) * D^{(k)}$
16: **end if**
17: $J^{(k)} \leftarrow |D^{(k)} - D_L^{(k)}|$
18: **if** $J_U^{(k-1)} < J^{(k)}$ **then**
19: $J_U^{(k)} \leftarrow J^{(k)}$
20: **else**
21: $J_U^{(k)} \leftarrow f^{(k)} * J_U^{(k-1)} + (1 - f^{(k)}) * J^{(k)}$
22: **end if**
23: **end event**

Se o tempo máximo de detecção definido pelo usuário é aproximadamente igual à estimativa do atraso de interação mínimo (i.e. $TD_U \approx D_L$), então o fator de esquecimento f tende a 0 (esquecimento máximo), assim o gestor autonômico não memoriza os valores de D_L, D_U e J_U e os mesmos variarão a cada observação. Note que, neste caso, o gestor autonômico não tem liberdade para a regulação do período de monitoramento – uma vez que, períodos de monitoramento muito menores que D_L podem implicar em um uso demasiado de recursos do ambiente, enquanto que períodos de monitoramento maiores que D_L violam o tempo máximo de detecção definido pelo usuário. Por outro lado, se o tempo máximo de detecção definido pelo usuário é muito maior que a estimativa do atraso de interação mínimo (i.e. $TD_U >> D_L$), então o fator de esquecimento f tende a 1 (esquecimento nulo), significando que a atualização dos valores de D_L, D_U e J_U não é relevante e os mesmos serão atualizados de forma mais conservadora pelo gestor autonômico (i.e. desconsiderando o fator de esquecimento). Neste caso, o gestor autonômico possui liberdade para ajustar o valor do período de monitoramento de modo a atender a relação de compromisso, da detecção de defeitos, entre custo (i.e. suposto percentual de consumo de recursos), velocidade (i.e. tempo de detecção) e precisão (i.e. quantidade e duração das falsas suspeitas).

Uma última questão a ser considerada pelo gestor autonômico na tarefa de sensoriamento do ambiente são os ajustes desnecessários no período de monitoramento, provocados por variações espúrias (ou repentinas) nos atrasos fim-a-fim. Observe que, apesar do período de monitoramento usado pelo serviço de detecção de defeitos contribuir para o suposto percentual de consumo de recursos (observado a partir dos atrasos), as aplicações que compõem o ambiente computacional possuem uma forte influência no atraso fim-a-fim, fazendo com que o mesmo possa variar aleatoriamente – incluindo, desse modo, vários picos espúrios (ou perturbações), provocadas, por exemplo, pelo início de uma nova transmissão ou chegada de um novo processo ao sistema.

Nesse sentido, usa-se de um filtro para evitar que o gestor autonômico ajuste desnecessariamente o período de monitoramento. Desse modo, após encontrar D_L, D_U e J_U, o gestor autonômico realiza uma filtragem em D de modo a eliminar variações espúrias. Assim, sendo D_F a versão filtrada de D, com $D_F^{(0)} = D^{(0)}$, calcula-se:

$$D_F^{(k)} = f^{(k)} * D_F^{(k-1)} + (1 - f^{(k)}) * D^{(k)}$$

As variáveis D, D_L, D_U, D_F representam informações do passado, obtidas a partir das mensagens de *heartbeats* recebidas. Para predizer o valor do atraso de interação, dito D_E, uma margem de segurança (J_U) é adicionada à versão filtrada de D:

$$D_E^{(k)} = D_F^{(k)} + J_U^{(k)}$$

A variável D_E é usada durante a regulação (ou sintonia) do período de monitoramento. Mais adiante, na descrição das estratégias de ajuste de *timeout* de detecção e de período de monitoramento, será apresentada uma discussão mais detalhada do uso de D_E e das demais variáveis estimadas e coletadas durante o sensoriamento do ambiente.

Sensoriamento da QoS de Detecção. O sensoriamento da qualidade do serviço básico de detecção é um aspecto chave para que o gestor autonômico possa compatibilizar o desempenho do serviço de detecção de defeitos com o desempenho demandado pelas aplicações. Para tanto, consiste em estimar, em tempo de execução, o desempenho do serviço de detecção em termos de: (a) *tempo de resposta*, o qual é representado pelo tempo de detecção (TD); e (b) *confiabilidade*, a qual é representada pela duração da falsa suspeita (TM), pelo intervalo entre falsas suspeitas (TMR) e pela disponibilidade de detecção (AV) – ver Seção 2.2.3 para uma descrição mais detalhada de TD, TM e TMR. A disponibilidade de detecção é uma métrica operacional introduzida em Sá e Macêdo (2010b), e é discutida mais adiante.

O sensoriamento do tempo de detecção é de particular interesse, pois é um elemento que permite ao gestor autonômico conciliar, durante a regulação do período, o

custo computacional associado ao suposto consumo de recurso do serviço de detecção com a velocidade da detecção. Todavia, uma estimativa mais precisa do tempo de detecção demanda uma análise estatística das falhas e certo conhecimento a respeito do ambiente de execução – o que pode não ser apropriado, uma vez que o ambiente possui características dinâmicas e, nesses casos, o de uso funções de distribuições de probabilidades adequadas não é uma tarefa simples. Sendo assim, a implementação do gestor considera como alternativa uma análise do tempo de detecção supondo que a falha acontece em um cenário de pior caso, isto é: um nó monitorado p_j falha imediatamente depois de enviar um *heartbeat* para um nó monitor p_i. Neste caso, p_i suspeitará de p_j quando alcançar o *timeout* de detecção do ciclo de monitoramento seguinte (ver Figura 3.10).

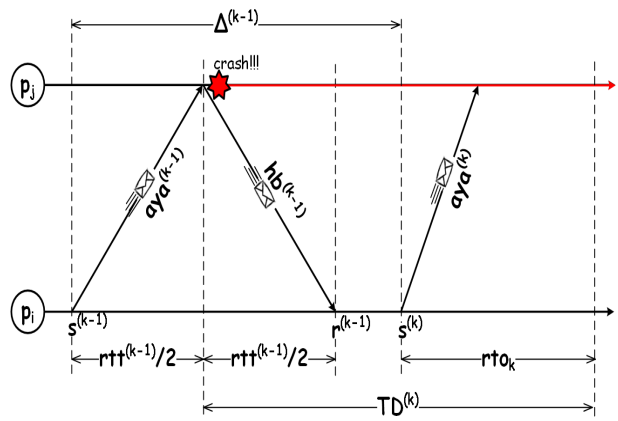


Figura 3.10: Cenário de pior caso para a estimativa de TD

Usando essa hipótese, o gestor autonômico estima o tempo de detecção da falha de p_j, usando $t_{crash}^{(k)} = r^{(k-1)} - rtt^{(k-1)}/2$, em que t_{crash} representa o suposto instante da falha do processo p_j. Assim, p_i suspeitará da falha p_j em $t_{suspect}^{(k)} = s^{(k)} + TO^{(k)}$. Com isso, estima-se o tempo de detecção:

$$TD^{(k)} = t_{suspect}^{(k)} - t_{crash}^{(k)} \tag{3.8}$$

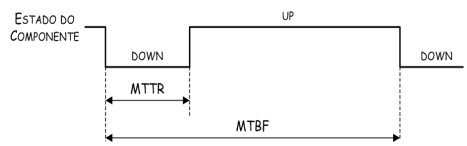
A Equação 3.8 representa um modo simples para estimar o tempo de detecção – a partir do qual gestor autonômico também não precisa utilizar qualquer conhecimento a priori sobre o ambiente ou sobre o comportamento, em termos de estatísticas de falhas, do processo que está sendo monitorado.

O sensoriamento da confiabilidade é importante para que o gestor autonômico con-

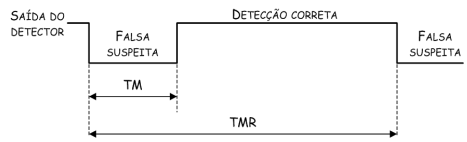
cilie, durante a regulação do *timeout* de detecção, a relação de compromisso entre velocidade e a confiabilidade do serviço de detecção. Nesse sentido, o gestor precisa garantir *timeouts* curtos, mas ao mesmo tempo assegurar a confiabilidade do detector mediante a imprecisão das estimativas do estimador de *timeout* quando condições dinâmicas do ambiente são consideradas.

Note que, o intervalo entre falsas suspeitas (TMR) e a duração das falsas suspeitas caracterizam de formas distintas a confiabilidade do detector. Enquanto TMR verifica a capacidade do detector em evitar falsas suspeitas (i.e. manter detecções livre de erros), TM avalia a capacidade do detector em corrigir rapidamente as falsas suspeitas cometidas (i.e. se recuperar de erros durante a realização das detecções) – é importante conciliar essas duas medidas de modo a facilitar a atuação do gestor autonômico.

Fazendo um analogia entre TMR e TM e as métricas usadas na análise da confiabilidade de sistemas, é possível observar que TMR e TM se assemelham, respectivamente, ao tempo médio entre defeitos (i.e. $MTBF$, *Mean Time Between Failures*) e ao tempo médio de reparo (i.e. $MTTR$, *Mean Time To Repair*), ver Figura 3.11.



(a) Métricas de confiabilidade de componente



(b) Métricas de confiabilidade de detecção

Figura 3.11: Analogia entre (TMR, TM) e ($MTBF$ e $MTTR$)

Na análise da confiabilidade, uma métrica que concilia ambas as métricas de confiabilidade é a disponibilidade. Assim, Sá e Macêdo (2010b) introduzem o conceito de disponibilidade de detecção:

Definição 3.3 ***Disponibilidade de Detecção*** *(AV): é uma métrica operacional da confiabilidade do detector que avalia a capacidade de um módulo monitor do serviço de detecção de defeitos em, quando requisitado, entregar informações livres de falsas suspeitas.*

Em Chen, Toueg e Aguilera (2002) é definida uma métrica similar, denominada

Probabilidade de consulta correta (PA, *query accuracy probability*), entretanto, essa métrica demanda um conhecimento a priori da função de distribuição de probabilidade para que a mesma seja calculada corretamente. A disponibilidade de detecção, por sua vez, é uma métrica operacional, sendo dessa forma uma métrica que avalia o desempenho do serviço em tempo de execução. A rigor, TM e TMR também precisam de distribuições de probabilidade definidas para serem calculadas. Entretanto, por conta de uma coerência com o jargão adotado na literatura, a denominação dessas métricas são mantidas – contudo, em (SÁ; MACÊDO, 2010b), utiliza-se uma metodologia diferente, da definida por Chen, Toueg e Aguilera (2002), e que permite que o cálculo de TM e TMR em tempo de execução. A seguir é discutido como a disponibilidade de detecção pode ser obtida.

A disponibilidade de um componente é determinada por:

$$AV = \frac{MTBF - MTTR}{MTBF}$$

De forma análoga, em um dado intervalo k, a disponibilidade média do módulo básico do serviço de detecção pode ser calculada pelo gestor autonômico usando:

$$AV^{(k)} = \frac{TMR^{(k)} - TM^{(k)}}{TMR^{(k)}} \tag{3.9}$$

O problema então é definir como usar a analogia adotada entre a disponibilidade do componente e a disponibilidade de detecção para obter TM e TMR e, dessa forma, calcular AV. Para tanto, observa-se que uma falsa suspeita pode ser comparada a um defeito de um módulo do serviço de detecção. Além disso, a cada intervalo k, o gestor autonômico calcula a duração de uma falsa suspeita ($SD^{(k)}$) e o número de falsas suspeitas ($NF^{(k)}$) por: $SD^{(k)} = r^{(k)} - (s^{(u)} - rtt^{(u)}/2)$ e $NF^{(k)} = NF^{(k-1)} + 1$, quando uma suspeita acontece; e $SD^{(k)} = 0$ e $NF^{(k)} = NF^{(k-1)}$, caso contrário. A variável $s^{(u)}$ representa o instante de envio do último $aya^{(u)}$, para o qual um respectivo $hb^{(u)}$ não foi recebido por um processo monitor p_i dentro do *timeout* de detecção $TO^{(u)}$.

As variáveis SD e NF podem ser associadas ao tempo de reparo de um defeito no serviço de detecção e ao número de reparos realizados no mesmo. Assim, associando o valor médio de TM ao tempo médio para reparo $MTTR$, o gestor autonômico pode estimar TM em um dado intervalo k por:

$$TM^{(k)} = \frac{1}{NF^{(k)}} \left(\sum_{x=0}^{k} SD^{(x)} \right) \tag{3.10}$$

Se até o instante $s^{(k)}$, um módulo monitor cometeu $NF^{(k)}$ falsas suspeitas, então o intervalo médio entre falsas suspeitas (ou o $MTBF$ do detector) é estimado pelo gestor autonômico usando:

$$TMR^{(k)} = \frac{s^{(k)} - s^{(0)}}{NF^{(k)}} \tag{3.11}$$

Desse modo, usando equações 3.10, 3.11 e 3.9 a disponibilidade de detecção pode ser encontrada. Assim, para conciliar os requisitos de confiabilidade de detecção definidas pelo usuário (i.e., TM_U e TM_L), Sá e Macêdo (2010b) definem o conceito de disponibilidade mínima de detecção:

Definição 3.4 ***Disponibilidade mínima de detecção*** *(AV_L): é uma métrica de confiabilidade do detector que determina a menor expectativa do usuário em termos de disponibilidade de detecção, podendo ser calculada da seguinte forma:*

$$AV_L = \frac{TMR_L - TM_U}{TMR_L} \tag{3.12}$$

Observe que, a disponibilidade de detecção depende das magnitudes dos intervalos entre falsas suspeitas e das durações das falsas suspeitas. Assim, usando como base a Equação 3.9, quando o intervalo entre falsas suspeitas diminui, a disponibilidade de detecção também diminui. Da mesma forma, quando a duração da forma suspeita aumenta, a disponibilidade de detecção diminui. Uma vez que as métricas TMR_L e TM_U representam, respectivamente, a expectativa do usuário em termos do intervalo mínimo entre falsas suspeitas e da duração máxima das falsas suspeitas, esses limites implicam na disponibilidade mínima esperada pelo usuário – conforme definido na Equação 3.12.

Regulação do Timeout de Detecção. Na detecção de defeitos em sistemas distribuídos, o *timeout* de detecção é um elemento chave, pois determina o prazo para que o nó monitorado responda antes que se torne suspeito de falha (ver Seção 2.2). Além disso, em ambientes com atrasos variados, qualquer estratégia de ajuste de *timeout* de detecção deve atender à relação de compromisso entre *velocidade* e *precisão* na detecção de defeitos. Isto é, *timeouts* de detecção (i.e. prazos) muito longos diminuem a ocorrência de falsas suspeitas, mas implicam em altas latências de detecção. Por outro lado, *timeouts* muito curtos, implicam em baixas latências de detecção, mas aumentam o número de falsas suspeitas.

Evidentemente que, em um ambiente com características dinâmicas e atrasos variados, qualquer estimador de atraso, usado no ajuste dos *timeouts* de detecção, come-

terá erros – seja sugerindo estimativas maiores que o atraso observado (prejudicando o tempo de detecção), seja sugerindo estimativas menores que o atraso observado (prejudicando a confiabilidade do detector). Com isso, as abordagens de detecção de defeitos, existentes na literatura, vêm privilegiando o atendimento ao requisito de confiabilidade, uma vez que, na grande maioria dos casos, as estratégias de ajuste de *timeout* usam estimadores que se adaptam as variações do atraso, mas utilizam alguma margem de segurança para prevenir falsas suspeitas – o que prejudica a velocidade da detecção, ver, por exemplo, Macêdo (2000), Chen, Toueg e Aguilera (2002), Bertier, Marin e Sens (2002), Falai e Bondavalli (2005), Satzger e outros (2007) etc. É razoável beneficiar a confiabilidade do detector, pois falsas suspeitas podem implicar em altos custos computacionais, oriundos da ativação de procedimentos de reconfiguração ou de recuperação, que demandam processamento e troca de mensagens adicionais.

Para reduzir o prejuízo ao tempo de detecção, alguns trabalhos têm usado margens de segurança adaptativas, em que a adaptação é realizada de acordo com as variações do atraso.

A abordagem de Jacobson (1988), por exemplo, usa uma estratégia baseada em médias móveis para estimar (ambos) o atraso e sua variação. Então, a mesma ajusta o *timeout* de detecção como sendo a média do atraso mais duas vezes sua variação[8] – o que pode garantir uma taxa de acerto de cerca de $95,45\%$, contando ainda com latências de detecção menores em momentos de estabilidade (isto é, nos quais as variações do atraso são menores).

A mesma consideração vale para estratégias como a de Bertier, Marin e Sens (2002), por exemplo. Esta estratégia também trabalha com média móvel (para a estimativa do atraso e da margem de segurança), considerando, entretanto, uma mobilidade de média menor que a usada em Jacobson (1988). Mais precisamente, enquanto Bertier, Marin e Sens (2002) consideram o histórico dos últimos 1000 atrasos, Jacobson (1988) baseia sua estimativa observando os dois últimos atrasos (ver Seção 2.3).

Note que o tamanho do histórico (i.e. a mobilidade da média) determina a velocidade com que a estimativa converge, acompanhando as variações dos atrasos. Quando as características do ambiente podem mudar, o desafio dessas estratégias é encontrar uma mobilidade de média que permita uma convergência adequada, atendendo ao compromisso entre velocidade e precisão das detecções. Isto é, se a estratégia confia em um histórico muito longo de atrasos (i.e. baixa convergência), a mesma pode demorar muito para responder às mudanças nas características do atraso (podendo, a depender

[8]Estatisticamente, se os dados são normalmente distribuídos, a média mais duas vezes o desvio padrão abrange $95,45\%$ dos dados – isto é conhecido como uma *regra de ouro*, nomeada *regra dos três sigma* ou *regra "*68-95-$99,7$*"*, ver Dai e Wang (1992).

da situação, prejudicar a velocidade ou a confiabilidade do detector)[9]. Por outro lado, se a estratégia confia em um histórico limitado de atrasos (i.e. convergência alta), a mesma estará mais suscetível a variações esporádicas do atraso (podendo cometer um maior número de falsas suspeitas) – como é verificado em muitos experimentos que usam o estimador de Jacobson (1988) para implementar o detector adaptativo.

Mais ainda, tentar estimar as características (i.e. distribuições) dos atrasos em tempo de execução, como é feito em Dixit e Casimiro (2010), por exemplo, pode implicar em soluções complexas e que, em alguns casos, desperdiçam recursos computacionais e não detectam a distribuição – ou quando detectam, a distribuição detectada não corresponde mais a distribuição que caracteriza o comportamento do ambiente.

O Algoritmo Proposto para a Regulação de Timeout de Detecção. A proposta de regulação de *timeout*, apresentada em (SÁ; MACÊDO, 2010b), não apenas aproveita os benefícios trazidos pelas abordagens de adaptação de *timeout* existentes, mas também realiza correções nos *timeouts* sugeridos, considerando as mudanças dinâmicas nas características do ambiente e nos requisitos definidos pelo usuário. Para isto, esta proposta de regulação usa um controlador integral que monitora a confiabilidade do detector, a partir da disponibilidade de detecção. Então, sugere uma margem de segurança a ser adicionada ao *timeout* de detecção, de modo a atender à disponibilidade mínima definida pelo usuário, com um menor prejuízo para o tempo de detecção. Mais precisamente, se o detector é impreciso (i.e. a disponibilidade AV é menor que a mínima), então a margem de segurança SM é incrementada para tornar a detecção mais precisa. Por outro lado, se a disponibilidade AV é alta (disponibilidade observada maior que a disponibilidade mínima), então a margem de segurança SM é decrementada para tornar a detecção mais rápida.

Para não comprometer o desempenho do detector, é importante estabelecer os limites para a margem de segurança (SM) sugerida pelo regulador de *timeout*. No estilo *Pull* (ver Seção 3.3.1), o menor *timeout* de detecção (i.e. prazo) possível é o atraso de comunicação mínimo (D_L). Então, SM deve ser menor ou igual a $TD_U - D_L$, de modo a não extrapolar o tempo máximo de detecção (TD_U) definido pelo usuário. Além disso, SM deve ser maior ou igual a zero, para não induzir o detector a cometer falsas suspeitas.

O Algoritmo 3.4 apresenta o procedimento usado na regulação do *timeout* de detecção. Antes do envio de cada mensagem de monitoramento, a disponibilidade mínima (AV_L) e a disponibilidade atual ($AV^{(k)}$) do detector são calculadas (Linhas 3–4). En-

[9]Neste caso, a confiabilidade da detecção é prejudicada quando o comportamento dos atrasos muda de uma variabilidade menos acentuada para uma variabilidade mais acentuada; e prejudica o tempo de detecção, caso contrário.

Algorithm 3.4 Regulação do *timeout* de detecção

1: $SM^{(0)} \leftarrow 0$
2: **before event** $aya^{(k)}$ sending:
3: $\quad AV_L \leftarrow \dfrac{TMR_L - TM_U}{TMR_L}$
4: $\quad AV^{(k)} \leftarrow \dfrac{TMR^{(k)} - TM^{(k)}}{TMR^{(k)}}$
5: $\quad error^{(k)} \leftarrow AV_L - AV^{(k)}$
6: $\quad SM^{(k)} \leftarrow SM^{(k-1)} + \Delta^{(k-1)} * error^{(k)}$
7: $\quad$ **if** $SM^{(k)} < 0$ **then**
8: $\quad\quad SM^{(k)} \leftarrow 0$
9: $\quad$ **end if**
10: $\quad$ **if** $SM^{(k)} > TD_U - D_L$ **then**
11: $\quad\quad SM^{(k)} \leftarrow TD_U - D_L$
12: $\quad$ **end if**
13: $\quad$ obtain $TO_C^{(k)}$ from the adaptive failure detector
14: $\quad TO^{(k)} \leftarrow TO_C^{(k)} + SM^{(k)}$
15: **end event**

tão, baseado no desvio ($error$) entre AV_L e $AV^{(k)}$, o regulador calcula a ação de controle integral (Linhas 5–6), isto é: $SM^{(k+1)} = SM^{(k)} + \Delta^{(k-1)} * error^{(k)}$.

Em seguida, o regulador limita a margem de segurança entre seus valores máximo e mínimo (Algoritmo 3.4, Linhas 7–11). Então, uma estratégia de adaptação de *timeout* é ativada para estimar o *timeout* (TO_C) de acordo com o atraso observado (Linha 13). Por fim, o procedimento define o *timeout* de detecção (TO) a ser usado pelo detector, i.e. a soma do *timeout* (TO_C), sugerido a partir dos atrasos, com a margem de segurança (SM), obtida a partir da disponibilidade (Linha 14). Note que o mecanismo de regulação de *timeout* não faz restrições com relação à estratégia de adaptação de *timeout*, baseada em atraso, que será usada para determinar TO_C. Nesse sentido, o regulador de *timeout* encapsula a estratégia de adaptação de *timeout* usada por um detector adaptativo.

Regulação do período de monitoramento. Na detecção de defeitos em sistemas distribuídos, o período de monitoramento é importante para determinar o intervalo entre as verificações do estado dos nós. Em ambientes com condições de carga variadas, o período de monitoramento deve ser escolhido de modo a atender à relação de compromisso entre custo computacional e desempenho do detector de defeitos.

Períodos de monitoramento muito longos reduzem o custo da detecção, uma vez que menos mensagens de monitoramento são transmitidas e processadas. Por outro lado, isto implica em detecções mais lentas e menos confiáveis. É evidente que quanto

mais longo o período, menor é o número de falsas suspeitas, pois menos vezes o detector de defeitos realiza suposições a respeito do estado do processo monitorado. Entretanto, neste contexto, a detecção dos defeitos se baseia em uma informação de estado muito envelhecida e menos provável de corresponder ao estado mais recente. Períodos de monitoramento muito curtos podem elevar demasiadamente o custo computacional das detecções, podendo exaurir os recursos disponíveis e implicar em atrasos mais longos e variados – o que compromete o tempo de resposta das aplicações e diminui a eficiência dos detectores e dos demais mecanismos de tolerância a falhas.

Os períodos de monitoramento também comprometem a qualidade do ajuste dos *timeouts* de *detecção*. As estratégias de adaptação de *timeout* precisam de amostras dos atrasos de comunicação (ou de outra informação do ambiente) para realizar suas estimativas. Portanto, menos precisos são os *timeouts* estimados, quando períodos longos são usados – uma vez que menos amostras são coletadas.

Assim, a escolha de períodos de monitoramento apropriados demanda um conhecimento dos recursos disponíveis (MILLS et al., 2004) ou algum conhecimento sobre a distribuição de carga de trabalho das aplicações (CHEN; TOUEG; AGUILERA, 2002) – o que é um desafio em ambientes distribuídos dinâmicos ou abertos. Isto é, em ambientes abertos é muito difícil determinar a quantidade de recursos disponíveis em um dado instante, como ocorre em Mills e outros (2004). Além disso, se as características do ambiente mudam, é muito difícil determinar a quantidade de recursos disponível a partir de distribuições de probabilidade específicas, como é feito em Chen, Toueg e Aguilera (2002).

As abordagens de Regulação de Período. A regulação de período de monitoramento, proposta em (SÁ; MACÊDO, 2010b), visa possibilitar detecções mais rápidas, confiáveis e com custo computacional adequado, considerando ambientes com características dinâmicas e requisitos de usuário que podem mudar.

Dada a dificuldade em se determinar com precisão os recursos disponíveis a cada instante, a proposta de regulação de período explora o atraso de interação fim-a-fim como uma metáfora para a determinação do percentual de uso (ou consumo) dos recursos disponíveis a cada instante. O conceito básico, associado a esta metáfora, é que quanto maior a carga das aplicações, dos mecanismos de tolerância a falhas ou do próprio detector de defeitos, maior é o consumo dos recursos, significando maior troca de mensagens, maior processamento, mais uso de memória e, consequentemente, atrasos computacionais maiores. Do mesmo modo, uma mudança nas características do ambiente que leva a uma variação (i.e. aumento ou redução) do poder de processamento, da quantidade de memória ou da capacidade de transferência na rede, também tende

a implicar em variações no percentual do uso dos recursos disponíveis[10] e nos atrasos computacionais.

Mais especificamente, a regulação do período usa um controlador o qual observa as variações nos atrasos de interação entre os processos. Em seguida, baseado nas variações de atrasos observadas, usa uma relação previamente definida para estimar o percentual de consumo de recursos (RC) no ambiente. Então, a partir do desvio entre os percentuais de consumo de recursos estimado e o desejado pelo usuário, o controlador usa uma lei de controle para determinar que variação deve ser realizada no período de monitoramento. Esse ajuste no período é realizado de modo a adequar o consumo de recursos do detector à fatia de recursos disponível no ambiente em um dado instante – buscando, assim, o menor período que não interfira no desempenho das aplicações ou dos demais mecanismos de tolerâncias a falhas.

Para atender ao objetivo proposto, o gestor autonômico implementa um laço de controle que considera três atividades: (i) *caracterização do consumo de recursos no ambiente computacional*, refere-se a atividade de estabelecer relações que ajudem o gestor autonômico a inferir o consumo de recursos no ambiente em um dado instante – uma vez que os atrasos das mensagens de monitoramento são a única informação disponível para o gestor autonômico, deve-se estabelecer uma relação entre tais atrasos e o consumo de recursos e deste último com o período de monitoramento; (ii) *definição da lógica do laço de controle*, é uma atividade na qual se determina quais ações de controle (ou ajustes) devem ser realizadas sobre o período de monitoramento quando o consumo de recursos no ambiente se afasta do desejado; e (iii) *projeto e sintonia do controlador*, é uma atividade relacionada à definição do algoritmo de controle e seus respectivos parâmetros.

Uma vez que se assume que as características do ambiente são desconhecidas e mudam dinamicamente, o projeto do mecanismo de regulação de período trata o ambiente como um sistema em caixa preta e usa as variações nos atrasos (i.e., D, D_L, D_U e J_U) como uma estimativa para o percentual do consumo de recursos do ambiente computacional.

Apesar de os atrasos poderem variar de forma não determinística, a proposta de regulação de período usa equações lineares para modelar o consumo de recursos e estimar o relacionamento entre o atraso e o período de monitoramento. Esse modelo baseado em equações lineares é uma aproximação e não descreve de forma apropriada o problema da caracterização do comportamento do ambiente computacional em

[10]Note que, se a carga computacional se mantém constante, mas acontece uma variação na quantidade nominal dos recursos disponíveis, também existirá uma variação relativa no percentual de consumo de recursos.

termos de consumo de recursos. Entretanto, tal modelo é uma boa ferramenta para a descrição das questões relacionadas ao problema de controle, à definição do comportamento dinâmico do ambiente distribuído e ao projeto da lei de controle. Para contornar as limitações impostas pela modelagem baseada em equações lineares, foi projetada uma lei de adaptação para a sintonia dos parâmetros do modelo, a qual permite que o modelo considerado se ajuste às mudanças nas características do ambiente computacional.

Para a regulação de período de monitoramento do detector de defeitos, duas propostas foram implementadas, usando as atividades e considerações descritas acima: (i) RBL (***R**egulation **B**ased on **L**ittle's Laws*), se baseia em algumas leis operacionais da teoria das filas propostas e originadas das leis de Little (1961) para modelar o comportamento dinâmico do ambiente computacional[11]; e (ii) RBS (***R**egulation **B**ased on **S**imple Linearization*), realiza a modelagem baseada na linearização do comportamento dos recursos percebidos a partir dos atrasos[12]. Os aspectos relacionados ao projeto e a implementação da proposta de regulação de período de monitoramento dita RBL são discutidos a seguir. Maiores detalhes sobre a proposta RBS podem ser encontrados em (SÁ; MACÊDO, 2010b).

A Proposta RBL: Regulação Baseada nas Leis de Little. A proposta de regulação de período RBL abstrai o ambiente como uma caixa preta e estima o que seria o número médio de mensagens em fila a partir dos atrasos fim-a-fim observados pelo processo de sensoriamento do gestor autonômico (ver Figura 3.12).

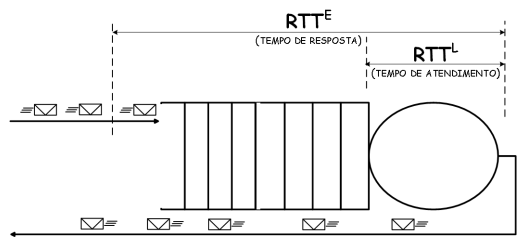


Figura 3.12: Modelo de filas suposto pelo gestor autonômico

O número médio de mensagens em fila ajuda o gestor autonômico a estimar o quanto de recurso está sendo consumido pelas aplicações que compartilham o ambiente em conjunto com o módulo do detector de defeitos que está sendo controlado. Para tanto, uma mensagem de monitoramento possui tamanho fixo e precisa de no mínimo $RTT_L = 2 * D_L$ unidades de tempo para ser processada pelo ambiente computacional

[11] A abordagem RBL, proposta neste trabalho, foi apresentada em (SÁ; MACÊDO, 2010a)
[12] A abordagem RBS, proposta neste trabalho, foi apresentada em (SÁ; MACÊDO, 2010b).

(i.e. envio de aya e recebimento de um hb), ver Figura 3.12. Se é observado um atraso de $RTT_E = 2 * D_E$ no processamento de uma mensagem de monitoramento (Figura 3.12), então, o número médio de mensagens sendo processados no sistema pode ser estimada, usando a lei de Little (1961), por:

$$M = \frac{RTT_E}{RTT_L} \tag{3.13}$$

Se quando um nó p_i insere uma mensagem no sistema, o mesmo observa um atraso médio (na resposta) equivalente a M mensagens, então essa mensagem esperou em fila $Q = M - 1$ mensagens serem processadas. Assim, usando a Equação 3.13, é possível obter:

$$Q = \frac{RTT_E}{RTT_L} - 1 \tag{3.14}$$

ou,

$$Q = \frac{RTT_E - RTT_L}{RTT_L} \tag{3.15}$$

Com isso, o percentual de utilização (RC) do sistema pode ser estimada usando:

$$RC = \frac{Q}{M_U} \tag{3.16}$$

em que M_U representa o número máximo de mensagens no sistema em um intervalo de observação T.

Uma vez que o tempo de resposta mínimo do sistema é RTT_L, então assumindo um intervalo de observação $T = \Delta$, estima-se que o sistema é capaz de processar:

$$M_U = \frac{\Delta}{RTT_L} \tag{3.17}$$

Assim, o percentual de utilização do sistema (RC) pode ser reescrita usando as equações 3.15 e 3.17:

$$RC = (RTT_E - RTT_L) * \lambda \tag{3.18}$$

em que $\lambda = 1/\Delta$ é a frequência de monitoramento.

Observe que a utilização, tal qual apresentada na Equação 3.18, não representa a utilização real, mas apenas uma estimativa vista pelo gestor autonômico instalado em um módulo monitor do detector de defeitos. Entretanto, tal informação serve como um indicativo para o consumo de recursos. Isto porque a Equação 3.18 consegue estabelecer uma relação entre a carga no ambiente a partir dos atrasos fim-a-fim estimados.

Isto é, possíveis variações na carga no ambiente computacional implicam em possíveis variações no atraso fim-a-fim estimado RTT_E, logo, segundo a Equação 3.18, o percentual consumo de recursos varia. Além disso, possíveis variações na freqüência de monitoramento implicam em variações na carga do detector, portanto o percentual de consumo de recursos também varia.

Usando a Equação 3.18, a taxa de variação (i.e. derivada) de RC em função de λ pode ser calculada por:

$$\frac{\partial RC}{\partial \lambda} = (RTT_E - RTT_L) \tag{3.19}$$

Do ponto de vista de um processo monitor p_i, se o gestor autonômico observa a variação de RC em função da variação de λ, então é possível calcular o relacionamento entre a entrada ($u = \partial\lambda$) e a saída ($y = \partial RC$) do sistema usando um modelo linear ARX[13]:

$$y^{(k+1)} = y^{(k)} + (RTT_E - RTT_L) * u^{(k)} \tag{3.20}$$

Observe que o efeito da variação da frequência de monitoramento em um intervalo k só tem efeito sobre a variação dos recursos, vistos pelo gestor autonômico em p_i, em $k + 1$.

O mecanismo de regulação de período observa a saída da planta y e atua sobre a entrada u e estas são as únicas variáveis vistas pelo mesmo. Entretanto, conforme ilustrado na Figura 3.13, desde a entrada (u) a até a saída (y) existem vários elementos envolvidos. De acordo com a Figura 3.13, mensagens de monitoramento são coletadas pelo elemento sensor, o qual extrai os atrasos D_L e D_E e então um dispositivo transdutor usa a Equação 3.18 para determinar o consumo de recursos.

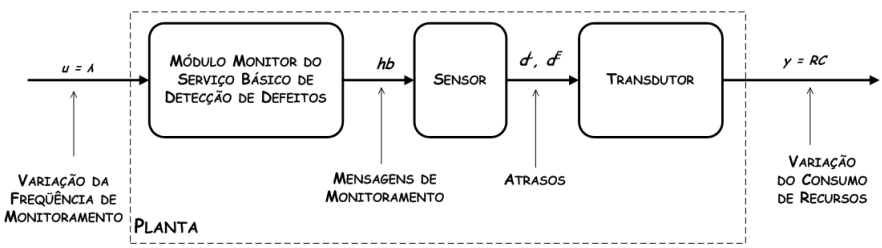


Figura 3.13: Diagrama em blocos da planta, em malha aberta, vista pelo regulador de período

[13] Ver Hellerstein e outros (2004) e Astrom e Wittenmark (1995) para mais detalhes sobre o uso do modelo ARX em teoria de controle.

O problema de controle é regular λ de modo a obter um menor tempo de detecção, usando uma fração do percentual de recursos disponíveis no ambiente a cada instante. Assim, seja $RC_D \in (0, 1]$ o percentual de recursos que pode ser consumida por um módulo monitor do detector de defeitos – em que RC_D é definido pelo usuário. Então, o percentual de consumo de recursos atual $RC^{(k)}$ é comparada a RC_D e a diferença $error^{(k)} = RC_D - RC^{(k)}$ é calculada. Quando $error^{(k)} > 0$, a utilização está abaixo do limite especificado, logo λ é incrementado para permitir uma detecção mais rápida. Se $error^{(k)} < 0$, a utilização está acima do limite desejado, logo λ é decrementado.

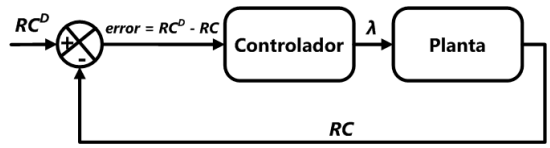


Figura 3.14: Laço de controle implementado pelo regulador RBL

A Figura 3.14 ilustra o laço de controle implementado pelo mecanismo de regulação RBL.

O gestor autonômico utiliza um controlador P (HELLERSTEIN et al., 2004), no qual a ação de controle (variação de λ) é proporcional ao desvio entre o valor desejado (RC^D) e o valor obtido (RC) para o consumo de recursos, isto é $\partial\lambda = Kp * (RC_D - RC^{(k)})$ – ou:

$$u^{(k)} = Kp * error^{(k)} \tag{3.21}$$

A sintonia do controlador diz respeito a encontrar o valor do ganho (ou parâmetro) proporcional Kp de modo a atender aos seguintes requisitos de desempenho (HELLERSTEIN et al., 2004): estabilidade; precisão; tempo de convergência (*settling time*); e máximo sobre-sinal (*overshoot*).

Uma vez que, o comportamento do ambiente muda, os requisitos de desempenho do controlador não podem ser garantidos com um controlador linear (P). Além disso, o projeto do controlador considera a escolha de intervalos fixos de tempo, dito período de amostragem, nos quais o controlador observa o valor do consumo de recursos e então atua sobre a freqüência de monitoramento (λ). Tais períodos de amostragem também não podem ser garantidos uma vez que a observação de RC em um dado instante depende do tempo de ida-e-volta de uma mensagem de monitoramento.

Mais ainda, o projeto tradicional baseado em funções de transferências descritas pela transformada $\mathcal{Z}$ é uma ferramenta válida apenas se o período de amostragem é fixo, portanto, qualquer análise utilizando funções de transferência em $\mathcal{Z}$ também não

será válida. Para contornar tal problema, utiliza-se uma configuração inicial do controlador P, usando funções de transferência em $\mathcal{Z}$, e então aplica-se uma lei de adaptação para o ganho Kp, de modo a atender ao menos o requisito de estabilidade. Sendo assim, a configuração inicial usa as equações 3.20 e 3.21 para obter as funções de transferências:

$$P(z) = \frac{RTT_E - RTT_L}{z-1} \quad e \quad C(z) = K_p$$

em que $P(z)$ e $C(z)$ representam o comportamento da planta e a lei de controle P, respectivamente.

Então, usa-se $P(z)$ e $C(z)$ para definir a função de transferência em malha fechada:

$$F_r(z) = \frac{C(z) * P(z)}{1 + C(z) * P(z)}$$

A estabilidade é garantida se $1 + C(z) * P(z) = 0$, ou seja: $z - 1 + Kp * (RTT_E - RTT_L) = 0$. Usando a técnica de localização dos pólos em malha fechada (HELLERSTEIN et al., 2004) e definindo um máximo sobre-sinal $M_P = 1 - RC_D$, é possível calcular:

$$K_P^{(k)} = \begin{cases} 1, & se \ RTT_E^{(k)} = RTT_L \\ \dfrac{RC_D}{RTT_E^{(k)} - RTT_L^{(k)}}, & se \ RTT_E^{(k)} > RTT_L \end{cases} \tag{3.22}$$

Sá e Macêdo (2010b) descrevem como a função de transferência para o modelo da planta é obtida e como o máximo sobressinal (M_P) e a lei de adaptação para K_P são encontrados.

Finalmente, O Algoritmo 3.5 apresenta o procedimento de regulação de período de monitoramento a partir da abordagem RBL. Nesse algoritmo 3.5, λ_L e λ_U representam as frequências de monitoramento mínima e máxima, respectivamente. Estes limites são usados para evitar que os valores das frequências determinem períodos de monitoramento que extrapolem os limites desejados. Tais limites são calculados a partir do tempo de detecção mínimo (TD_L) estimado e do tempo de detecção máximo (TD_U) especificado pelo usuário. Mais detalhes sobre os valores limites para λ são encontrados em (SÁ; MACÊDO, 2010b).

Algorithm 3.5 Regulação de período de monitoramento (proposta RBL)

1: **before event** $hb^{(k)}$ sending:
2: use the Equation 3.22 to compute $K_p^{(k)}$
3: use the Equation 3.18 to compute $RC^{(k)}$
4: obtain RC_D
5: $error^{(k)} \leftarrow RC_D - RC^{(k)}$
6: $\lambda^{(k)} \leftarrow \lambda^{(k-1)} + K_P * error^{(k)}$
7: **if** $\lambda^{(k)} < \lambda_L$ **then**
8: $\lambda^{(k)} \leftarrow \lambda_L$
9: **end if**
10: **if** $\lambda^{(k)} > \lambda_U$ **then**
11: $\lambda^{(k)} \leftarrow \lambda_U$
12: **end if**
$\Delta^{(k)} \leftarrow \frac{1}{\lambda^{(k)}}$
13: **end event**

3.3.3 Avaliação de Desempenho

Ambiente de Simulação. Os experimentos foram realizados por simulação com o auxílio do pacote *Matlab/Simulink/TrueTime 1.5* (HENRIKSSON; CERVIN, 2007). Para os experimentos, foram configurados três computadores, ditos c_1, c_2 e c_3, conectados através de uma rede *Switched Ethernet* com taxa nominal de transferência de $10Mbps$, *buffer* de $1MB$ e em caso de *buffer overflow* mensagens são descartadas. As mensagens trocadas entre os nós do ambiente têm tamanho fixo e igual a $1536\ bits$ – equivalente a três vezes o tamanho mínimo de um quadro *Ethernet*.

O processo em c_1 monitora defeitos do processo em c_2. Em c_3, um processo (denominado p_3) é utilizado para gerar rajadas aleatórias de trafego na rede. Essas rajadas são geradas de tal modo que a utilização média da rede é incrementada de 10% a cada $1000ms$ – sendo que a mesma retorna a zero após atingir 90%.

Para tanto, p_3 é ativado periodicamente a cada $0,1536ms$, i.e. 1536 bits dividido por $10Mbps$ – o que equivale ao tempo necessário para transmitir um quadro de 1536 bits. Uma variável $bw \in [0, 1)$ é mantida por p_3, sendo a mesma inicializada com zero, incrementada de $\frac{1}{10}$ a cada $1000ms$ e retorna à zero após atingir $0,9$. Uma função geradora de números pseudoaleatórios, dita *rand()* é chamada por p_3 a cada ativação. A função *rand()* retorna um valor uniformemente distribuído no intervalo $(0, 1)$. Um quadro é enviado por p_3 quando o valor retornado pela função *rand()* é menor que bw. Essa geração de rajadas aleatórias permite avaliar os detectores de defeitos sob diferentes condições de carga. Por fim, as simulações são executadas até que tenham sido transferidas 10^4 mensagens de monitoramento.

Métricas de desempenho. O desempenho dos detectores foram verificados considerando as métricas: *Tempo de Detecção* (TD), *Duração da Falsa Suspeita* (TM), *Intervalo entre Falsas Suspeitas* (TMR) e *Disponibilidade de Detecção* (AV), vide Seções 2.2.3. Além disso, utiliza-se também, como métrica de desempenho, a *Taxa de Falsas Suspeitas* (RM, *Rate of Mistakes*), a qual é obtida dividindo o número de falsas suspeitas (nf) pelo número total de verificações que o detector realiza a respeito do processo monitorado. Essa última métrica é mais justa que o uso do número de falsas suspeitas somente. Isto porque, detectores de defeitos com períodos de monitoramento mais longos verificam menos vezes o estado do processo monitorado. Portanto, tendem a cometer menos suspeitas por intervalo de tempo.

Configuração dos detectores. Nos experimentos realizados são considerados quatro detectores de defeitos: (1) Detector autonômico baseado na abordagem RBL, denominado $AFD\text{-}RBL$; (2) Detector autonômico baseado na abordagem RBS, denominado $AFD\text{-}RBS$; (3) Detector adaptativo baseado no estimador de *timeout* de Jacobson (1988), denominado $Jacobson$; e (4) Detector adaptativo baseado no estimador de *timeout* de Bertier, Marin e Sens (2002), denominado $Bertier$;

As abordagens de detecção autonômicas $AFD\text{-}RBL$ e $AFD\text{-}RBS$ usam um estimador de *timeout* e realizam correções nas estimativas realizadas pelo mesmo quando necessário. Assim, nos experimentos, ambas as abordagens autonômicas usam o estimador de *timeout* de Jacobson (1988) e as correções são realizadas usando os procedimentos apresentados na Seção 3.3.2 (ver Algoritmo 3.4). Além disso, $AFD\text{-}RBL$ e $AFD\text{-}RBS$ foram configurados da seguinte forma: $RC^D = 0,5$ (i.e. 50% do percentual dos recursos não utilizados pelas aplicações); $TD^U = 50ms$ (i.e. aproximadamente 20 vezes a latência mínima de transmissão de um quadro na rede); e $TM^U = 1ms$ e $TMR^L = 10000ms$ (i.e. $AV^L = 0,9999$).

Os detectores adaptativos, usando os estimadores de Bertier, Marin e Sens (2002) e de Jacobson (1988), foram configurados conforme originalmente proposto pelos autores. Nas comparações, consideram-se quatro configurações para os detectores adaptativos. Essas configurações usam período de monitoramento fixo e equivalentes a 1, 3 e 5 milissegundos, respectivamente. Esses valores de período de monitoramento foram selecionados experimentalmente, considerando os seguintes aspectos observados: (a) períodos de monitoramentos inferiores a $1ms$ foram utilizados, mas os mesmos apenas aumentaram a variabilidade dos atrasos na rede e o número de falsas cometidas pelos detectores adaptativos – sem trazer qualquer benefício para o tempo de detecção; (b) períodos superiores a $5ms$ apenas incrementaram o tempo de detecção, sem melhorar a precisão dos detectores adaptativos. Portanto, os períodos de monitoramento foram

fixados dentro da faixa de 1 a $5ms$ para garantir uma comparação mais justa entre as abordagens de detecção autonômica e as abordagens de detecção adaptativa com períodos pré-fixados.

Resultados obtidos. As figuras de 3.15 a 3.19 apresentam o desempenho dos detectores de defeitos em termos das métricas consideradas e em diferentes condições de carga. Nessas figuras, os eixos x e y dos gráficos representam o tempo em milissegundos e a métrica considerada, respectivamente. Nas figuras 3.15, 3.16 e 3.18, TD, TM e TMR são representados em milissegundos. Nas figuras 3.16 e 3.18, os eixos y, referentes a TM e TMR, estão em escala logarítmica. Os gráficos da Figura 3.19, os quais se referem ao desempenho em termos da disponibilidade de detecção, possuem eixos y com escalas diferentes.

Em todas as figuras, os gráficos intitulados Bertier($\delta = x\,ms$) referem-se ao detector adaptativo implementado com o algoritmo de Bertier, Marin e Sens (2002) e configurado com período de monitoramento (δ) igual a x milissegundos. Da mesma forma, os intitulados Jacobson($\delta = x\,ms$) referem-se ao detector adaptativo implementado com o algoritmo de Jacobson (1988) e configurado com período de monitoramento igual a x milissegundos.

Na Figura 3.15, referente ao desempenho em termos do tempo de detecção, todos os eixos y dos gráficos estão na mesma escala (variando entre 0 e $8\,ms$), com exceção dos gráficos pertinentes a Bertier($\delta = 1\ ms$) e Jacobson($\delta = 1\ ms$), os quais possuem eixo y variando de 0 a $80\,ms$ – pois os tempos de detecção dos mesmos foram muito superiores aos demais e, usar uma escala comum, em todos os gráficos, não permitiria que os demais resultados fossem comparados de forma adequada.

Por fim, ao final desta seção, a Tabela 3.3 apresenta um resumo dos desempenhos médios obtidos, pelos detectores, durante os experimentos realizados.

Em termos do tempo de detecção de defeitos, Figura 3.15, as configurações de detectores adaptativos de defeitos denominadas Bertier($\delta = 1\ ms$) e Jacobson($\delta = 1\ ms$) apresentam baixos tempos de detecção de defeitos para baixas condições de carga, mas elevam drasticamente o tempo de detecção de defeitos em cenários com cargas maiores que 70% – apresentando tempos médios de detecção defeitos de $2,7ms$ e $3,0ms$, respectivamente. Note que os detectores adaptativos de defeitos com $\delta = 1ms$ consomem $\approx 30,72\%$ da banda da rede (i.e., uso de rede durante $\frac{1536bits}{10Mbs}$ segundos vezes 2 mensagens a cada $1ms$). Isto significa que, nos intervalos $[7000ms, 9000ms]$, $[17000ms, 19000ms]$, $[27000ms, 29000ms]$ e $[37000ms, 39000ms]$, a demanda de recurso de rede é maior que 100% (o que justifica os picos nos valores de TD observados em tais intervalos para esses detectores).

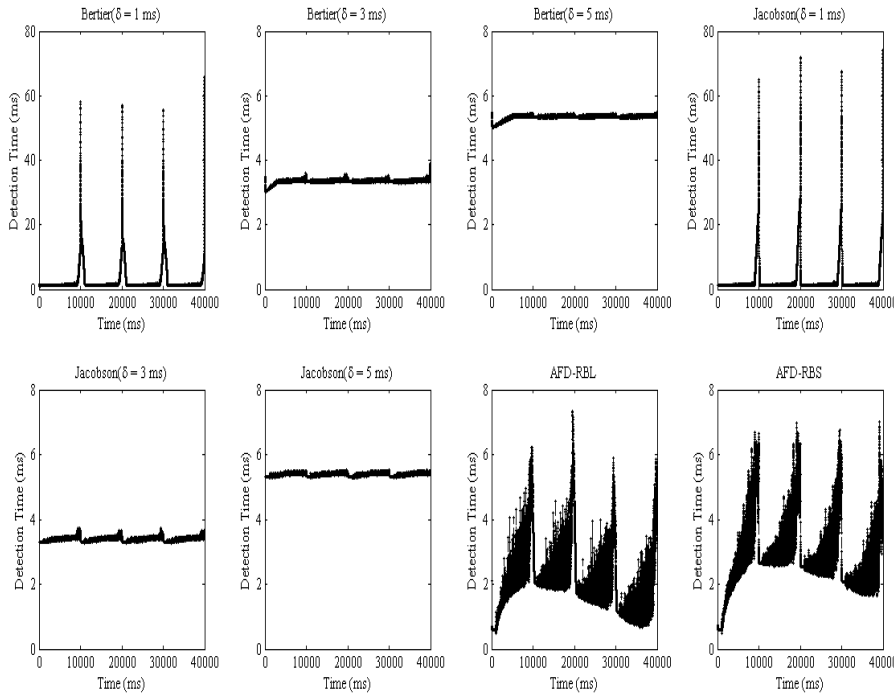


Figura 3.15: Desempenho em termos do tempo de detecção

Os detectores adaptativos de defeitos com $\delta = 3ms$ e $\delta = 5ms$, por sua vez, possuem tempos médios de detecção de defeitos muito próximos aos seus respectivos valores de período de monitoramento – i.e., $3,4ms$ e $5,3ms$ para Bertier($\delta = 3\,ms$) e Bertier($\delta = 5\,ms$) e $3,4ms$ e $5,9ms$ para Jacobson($\delta = 3\,ms$) e Jacobson($\delta = 5\,ms$).

Os detectores autonômicos de defeitos ($AFD\text{-}RBL$ e $AFD\text{-}RBS$) possuem tempos de detecção de defeitos muitos similares, sendo que estes tempos de detecção variam entre $0,5ms$ e $7,0ms$ dependendo da carga, apresentando valores médios de TD aproximadamente iguais a $1,7ms$ ($AFD\text{-}RBL$) e $2,5ms$ ($AFD\text{-}RBS$). Sendo assim, os detectores autonômicos $AFD\text{-}RBL$ e $AFD\text{-}RBS$ possuem desempenho médio em termos do tempo de detecção defeitos superior a todos os detectores adaptativos de defeitos considerados, tendo $AFD\text{-}RBL$ um desempenho médio um pouco melhor que $AFD\text{-}RBS$, em termos desta métrica.

O detector $AFD\text{-}RBS$ possui um tempo médio de detecção de defeitos superior ao $AFD\text{-}RBL$ por conta do efeito da ação integral usada pelo mecanismo de regulação de período RBS. Tal ação integral faz com que o detector defeitos memorize o efeito

das variações de carga passadas e sugira períodos de monitoramento um pouco mais elevados que a abordagem de regulação de período RBL.

Os resultados apresentados demonstram que o ajuste dinâmico do período de monitoramento, realizado pelas abordagens de detecção autonômicas, permite acomodar adequadamente a carga gerada pelo detector aos recursos disponíveis a cada instante. Além disso, estas abordagens permitem tempos de detecção mais curtos em cenários com menor carga de aplicação. Essas facilidades não são possíveis quando se usa períodos de monitoramento fixos.

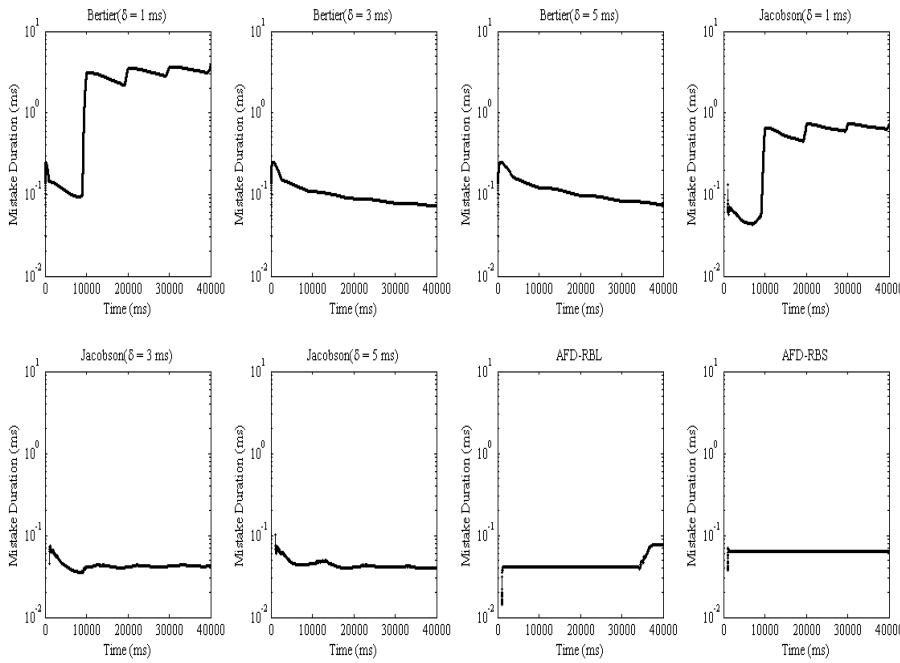


Figura 3.16: Desempenho em termos da duração da falsa suspeita

Em termos da duração das falsas suspeitas, Figura 3.16, os detectores adaptativos de defeitos com $\delta = 1ms$ apresentaram o pior desempenho, com TM médios de $2,40ms$ e $0,49ms$ para $Bertier$ e $Jacobson$, respectivamente. Mais ainda, o detector de defeitos Bertier($\delta = 1\,ms$), sob as condições de carga impostas, tem o seu desempenho prejudicado por conta do efeito de memória provocado pelo tamanho do histórico de mensagens utilizado em tal abordagem – i.e. $ws = 1000$, conforme originalmente proposto por Bertier, Marin e Sens (2002). Note que, nos intervalos de maior carga (i.e. $[7000ms, 9000ms]$, $[17000ms, 19000ms]$, $[27000ms, 29000ms]$ e

$[37000ms, 39000ms]$, respectivamente), esse detector de defeitos demora a perceber a mudança dos atrasos, pois a componente principal do *timeout* confia principalmente na média. Assim, no início desses intervalos, o mesmo dependerá basicamente do ajuste fino que é feito pela margem de segurança, calculada segundo a estratégia de Jacobson (1988) e que também sofre com a mudança de carga (ver Figura 3.15 e 3.16). Isto explica o comportamento observado no gráfico de TM para o detector adaptativo defeitos de $Bertier$ com $\delta = 1ms$.

Na medida em que o período de monitoramento cresce, a influência do mesmo sobre a variação da carga de trabalho na rede diminui. Assim, os estimadores de *timeout* podem realizar predições mais acuradas e, consequentemente, os detectores adaptativos de defeitos corrigem as suas falsas suspeitas mais rapidamente – observe que ambos os estimadores de Jacobson (1988) e de Bertier, Marin e Sens (2002) usam margens de segurança adaptativas para sobrestimar os atrasos, desse modo, se a variabilidade da carga diminui, a influência dessas margens de segurança é menor.

Com isso, os detectores adaptativos de defeitos com períodos de monitoramento $\delta = 3ms$ e $\delta = 5ms$ têm tempos de detecção de defeitos e durações de falsas suspeitas menores que aqueles com $\delta = 1ms$. Sendo assim, Bertier($\delta = 3\,ms$) e Bertier($\delta = 5\,ms$) apresentam valores médios de TM de $0,10ms$ e $0,11ms$, respectivamente. Enquanto que, a média de TM para Jacobson com $\delta = 3ms$ e $\delta = 5ms$ foi de $0,04ms$ para ambas as configurações, sendo, desta forma, melhor que o detector adaptativo de Bertier em termos de tal métrica.

Os detectores autonômicos de defeitos possuem períodos de monitoramento que se adaptam às diferentes condições de carga, de modo a minimizar o efeito de tal período de monitoramento sobre a carga da rede. Sendo assim, os mesmos induzem menos variações na carga, o que permite que os mesmos apresentem durações de falsas suspeitas mais estáveis. Com isso, os detectores autonômicos $AFD\text{-}RBL$ e $AFD\text{-}RBS$ apresentam durações média de falsas suspeitas de $0,04ms$ e $0,06ms$, respectivamente. Sendo que, conforme comentado anteriormente, o detector $AFD\text{-}RBS$, por conta da ação integral, responde mais lentamente, possuindo, assim, um desempenho médio um pouco inferior ao $AFD\text{-}RBL$ em termos de TM.

Os resultados apresentados demonstram que o ajuste dinâmico do período de monitoramento, realizado pelas abordagens de detecção autonômica de defeitos, permite acomodar adequadamente a carga gerada pelo detector de defeitos aos recursos disponíveis a cada instante. Consequentemente, estas abordagens também permitem durações de falsas suspeitas mais estáveis e curtos em cenários com menor carga de aplicação – o que não é observado nas abordagens de detecção adaptativas de defeitos com períodos de monitoramento fixos.

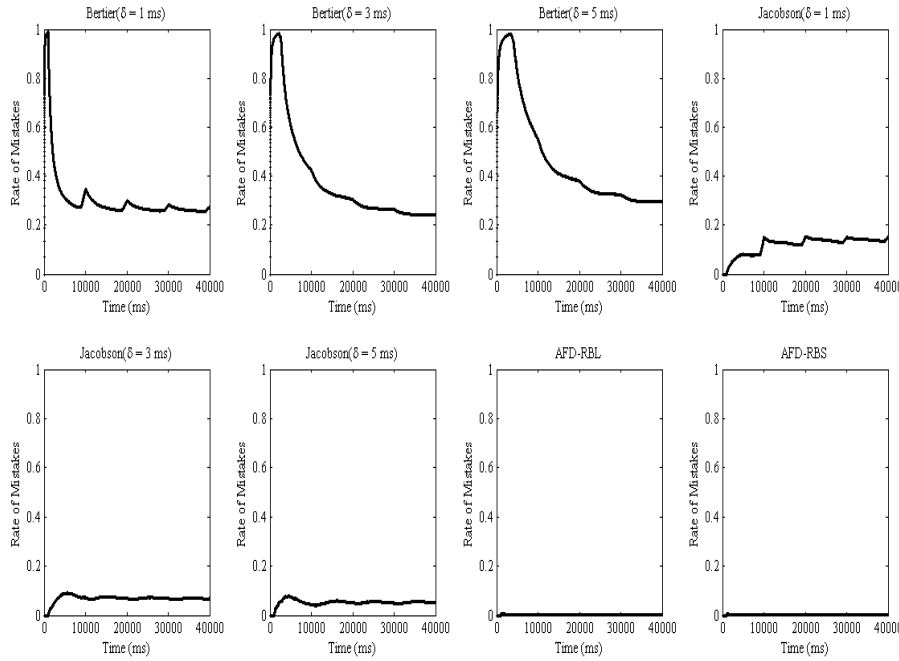


Figura 3.17: Desempenho em termos da taxa de falsas suspeitas

Em termos da taxa de falsas suspeitas, Figura 3.17, os detectores autonômicos apresentam um desempenho muito superior aos demais. Tais detectores apresentaram taxas de falsas suspeitas muito próximas de zero, mais precisamente $0,0013$ e $0,0007$ para $AFD\text{-}RBL$ e $AFD\text{-}RBS$, respectivamente.

O bom desempenho dos detectores autonômicos, em termos da taxa de falsas suspeitas, se deve a estratégia de ajuste da margem de segurança, combinada com o uso de períodos de monitoramento mais longos e dinamicamente calculados para altas condições de carga na rede. Tal combinação resulta em atrasos de comunicação mais estáveis. Os detectores adaptativos com o estimador de Bertier, Marin e Sens (2002) sempre apresentaram uma taxa de falsas suspeitas acima de $0,3$, para qualquer configuração considerada. Para tal detector, a taxa de falsas suspeitas média foi de $0,46$, $0,38$ e $0,30$ para δ igual a 1, 3 e $\delta = 5ms$, respectivamente. Os detectores adaptativos com o estimador de Jacobson (1988), por sua vez, apresentaram taxas de falsas suspeitas abaixo de $0,2$, para todas as configurações consideradas. Para tal detector, a taxa de falsas suspeitas média foi de $0,11$, $0,06$ e $0,05$ para $\delta = 1ms$, $\delta = 3ms$ e $\delta = 5ms$, respectivamente.

Em termos do intervalo entre falsas suspeitas, Figura 3.18, os detectores autonômicos de defeitos obtiveram o melhor desempenho quando comparados a todos os demais detectores adaptativos de defeitos considerados. As rampas apresentas no gráfico para os detectores autonômicos indicam que os mesmos levam muito mais tempo para cometer uma falsa suspeita.

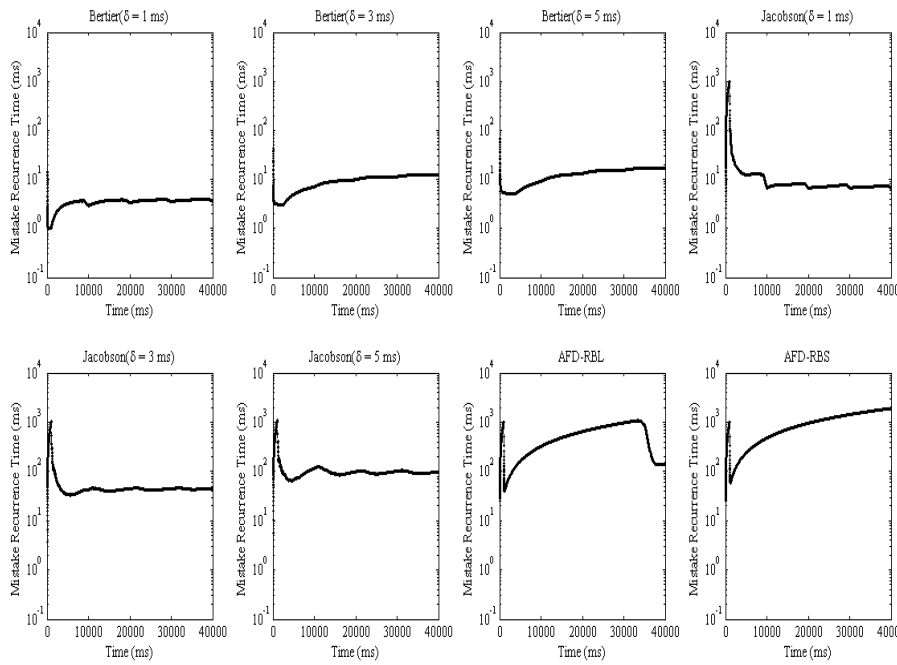


Figura 3.18: Desempenho em termos do intervalo entre falsas suspeitas

Além disso, os valores médios de TMR estão na ordem de 10^3 milissegundos, sendo os mesmos $548,78ms$ e $927,28ms$ para $AFD\text{-}RBL$ e $AFD\text{-}RBS$, respectivamente. O bom desempenho dos detectores autonômicos em termos de TMR também se deve à combinação da abordagem usada para o cálculo da margem de segurança combinada com maiores períodos de monitoramento dinamicamente calculados para altas condições de carga na rede.

Os detectores de defeitos $Jacobson$ obtiveram o segundo melhor desempenho em termos de TMR, sendo os valores médios de TMR iguais a $22,51ms$, $58,42ms$ e $109,70ms$ para períodos de monitoramento δ iguais a 1, 3 e $5ms$, respectivamente. Os detectores $Bertier$ teve o pior desempenho em termos de TMR quando comparado com os demais. Sendo que para tal detector de defeitos, as médias de TMR foram

$3,47ms$, $9,28ms$ e $12,55ms$, respectivamente. No caso dos detectores adaptativos de defeitos (i.e., $Jacobson$ e $Bertier$), o baixo desempenho se deve aos períodos de monitoramento fixados. E no caso dos detectores de $Bertier$, o desempenho muito ruim se também ao efeito de memória provocado pelo tamanho do histórico de mensagens utilizado pelo estimador de *timeout* usado por tal abordagem. Note que, o histórico longo faz com que o mesmo demore mais para responder a mudanças repentinas na carga, ocasionando um número maior de falsas suspeitas e, consequentemente menores valores médios de TMR.

Por fim, a Figura 3.19 apresenta o desempenho dos detectores em termos da disponibilidade de detecção.

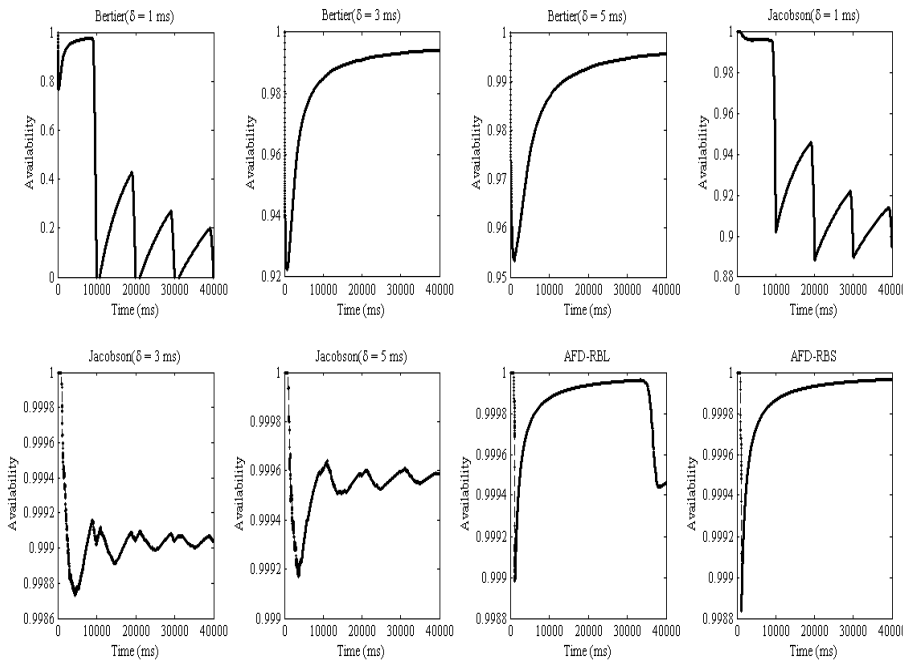


Figura 3.19: Desempenho em termos da disponibilidade de detecção

A disponibilidade do detector Bertier($\delta = 1\,ms$) decai até atingir valores de $0,3398$ na média, sendo este o pior desempenho em termo desta métrica. O detector $Bertier$ com $\delta = 3ms$ e $\delta = 5ms$ apresenta desempenho médio em termos de AV de $0,9849$ e $0,9881$, respectivamente. Ainda assim, o mesmo apresenta desempenho inferior aos detectores autonômicos e ao detector $Jacobson$. Observe que, no melhor caso o detector $Bertier$ apresenta disponibilidade média de dois 9 (i.e., $0,99$ aproximadamente).

O detector $Jacobson$ apresentaram disponibilidades médias de $0,9334$, $0,9991$ e $0,9995$ para δ igual a 1, 3 e $5ms$, respectivamente. Deste modo, o detector $Jacobson$ apresenta, no melhor caso, disponibilidade média de três 9 (i.e., $0,999$ aproximadamente).

Os detectores autonômicos $AFD\text{-}RBL$ e $AFD\text{-}RBS$ apresentaram disponibilidade média de $0,9998$ e $0,9999$, respectivamente. Sendo assim, os detectores autonômicos possuem desempenho em termos desta métrica muito melhor que os detectores adaptativos considerados. Observe que, no melhor caso, os detectores autonômicos possuem disponibilidade média de quatro 9 (i.e., $0,9999$ aproximadamente).

Nas figuras 3.16, 3.18 e 3.19, observa-se uma queda de QoS para $AFD\text{-}RBL$, no instante $35000ms$ das simulações. Isto porque, neste ponto, $AFD\text{-}RBL$ está com δ decrescendo (verifique a Figura 3.15), chegando ao ponto de interferência máxima (menor δ), até o ponto em que o período estabiliza e começa a aumentar como indicam os gráficos no fim da curva. O mesmo não acontece com $AFD\text{-}RBS$ por conta da ação integral empregada, por tal detector, no ajuste do período de monitoramento.

A Tabela 3.3 apresenta um resumo do desempenho médio dos detectores de defeitos em termo das métricas de qualidade de serviço de detecção consideradas nas avaliações.

Tabela 3.3: Desempenho médio dos detectores de defeitos em termos das métricas QoS de detecção consideradas

	Bertier et. al. (2002)			Jacobson (1988)			AFD-RBL	AFD-RBS
	$\delta = 1ms$	$\delta = 3ms$	$\delta = 5ms$	$\delta = 1ms$	$\delta = 3ms$	$\delta = 5ms$		
$TD\ (ms)$	2,69	3,35	5,34	3,04	3,39	5,90	1,72	2,54
$TM\ (ms)$	2,40	0,10	0,11	0,49	0,04	0,04	0,04	0,06
$TMR\ (ms)$	3,47	9,28	12,55	22,51	58,42	109,70	548,78	927,81
RM	0,46	0,38	0,30	0,11	0,06	0,05	0,00	0,00
AV	0,3398	0,9849	0,9881	0,9334	0,9991	0,9995	0,9998	0,9999

CAPÍTULO 4

Reconfiguração Dinâmica em Sistemas de Tempo Real

Eduardo Camponogara, Augusto Born de Oliveira e George Lima

Sistemas de tempo real podem ser informalmente definidos como aqueles que interagem sincronamente com o mundo real, percebendo nele a ocorrência de eventos e reagindo aos mesmos dentro de intervalos de tempo previamente estipulados. Esta interação é realizada através de uma interface composta de sensores e atuadores. Os sensores captam os eventos relevantes ao sistema em questão transformando-os em sinais de entrada. Estes são convertidos e armazenados em variáveis internas mantidas pelo sistema. Associados aos eventos captados pelos sensores pode haver uma ou mais tarefas, que devem ser executadas pelo sistema. Os resultados destas tarefas podem, por sua vez, modificar o estado do mundo real através dos atuadores.

Sistemas de tempo real estão presentes em diversos campos. Exemplos típicos podem ser encontrados na área de controle. Por exemplo, para manter a temperatura e pressão de uma caldeira industrial em níveis desejáveis, sensores de pressão e temperatura informam ao sistema computacional os valores medidos. Estes valores servem de entrada para o algoritmo de controle, que deve dar uma resposta num determinado intervalo de tempo, enviando um sinal para a válvula de combustível a fim de ajustar a temperatura e a pressão da caldeira.

O avanço tecnológico tem possibilitado o uso mais abrangente de sistemas de tempo real, em diversas aplicações nas áreas de robótica móvel, monitoramento com-

putadorizado, navegação assistida, *etc*. Nestes sistemas, sensores modernos capturam sinas de áudio e vídeo entre outros. Tais dados devem ser apropriadamente processados, respeitando restrições temporais. Os algoritmos de processamento, por sua vez, exibem grande variabilidade temporal, podendo sofrer influência de condições ambientais, variação de iluminação, ângulo de visão, nível de ruído do ambiente, *etc*. Tais cenários podem ser usados para definir diferentes *modos de operação*, cada um dos quais associados a condições de execução distintas. Por exemplo, dependendo das condições de iluminação, pode-se prever diferentes algoritmos de processamento de imagem, cada um provendo determinada qualidade de serviço e demandando mais ou menos recursos computacionais. Desta forma, suporte adequado a tais sistemas deve contemplar os seus requisitos adaptativos. Em outras palavras, deve-se prover suporte à *reconfiguração dinâmica* dos parâmetros de execução do sistema. Entende-se por reconfiguração dinâmica de um sistema a redistribuição, em tempo de execução, dos recursos computacionais entre as suas tarefas adequando seu comportamento às necessidades demandadas e preservando as restrições temporais correntes.

Este capítulo descreve alguns mecanismos para prover reconfiguração dinâmica para suporte a sistemas de tempo real adaptativos. Antes de abordá-los, alguns conceitos relacionados ao suporte à execução de tarefas em sistemas de tempo real serão apresentados. Mais especificamente, serão vistos mecanismos de escalonamento de tarefas que têm papel fundamental na garantia de correção temporal do sistema.

4.1 Escalonamento em Sistemas de Tempo Real

Escalonamento é um problema fundamental em sistemas de tempo real cujas soluções determinam a ordem de execução das tarefas do sistema. Como há prazos de execução que devem ser cumpridos pelas tarefas, ordenar a execução das mesmas têm implicações diretas na correção do sistema.

Para efeito de especificação do problema de escalonamento e de derivação de suas soluções, um sistemas de tempo real é comumente visto como um conjunto de tarefas $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, cada uma das quais definindo um conjunto de instruções a serem executadas. Cada tarefa $\tau_i \in \Gamma$ está associada a atributos temporais e a relações de precedência ou dependência. Tais relações exprimem, por exemplo, aspectos como compartilhamento de recursos ou restrições na ordem de execução das tarefas. Atributos temporais comuns são custo máximo de execução (C_i), prazo máximo de execução (D_i) e período (T_i). Se τ_i é uma tarefa periódica, instâncias de τ_i são liberadas para execução a cada T_i unidades de tempo. Caso τ_i seja uma tarefa esporádica, T_i signi-

fica a distância mínima entre duas liberações consecutivas de suas instâncias. Tarefas também podem ser aperiódicas e, neste caso, não há o atributo T_i ou ele representa a média de tempo entre ativações de τ_i. Em todo caso, se uma instância de τ_i é liberada para execução no instante t, o instante máximo para finalizar sua execução, conhecido como seu *deadline* absoluto, será $t + D_i$. O atributo D_i é conhecido como *deadline* relativo da tarefa τ_i.

Para os propósitos deste capítulo, assumimos que o *deadline* relativo das tarefas é igual aos seus respectivos períodos, e as tarefas são independentes uma das outras, ou seja, não há relações de precedência ou dependência entre as mesmas. Desta forma, uma tarefa τ_i pode ser definida como uma tupla, isto é, $\tau_i = (C_i, T_i)$. Além disto, assumimos que o sistema contém apenas uma unidade de processamento.

Resolver o problema de escalonamento em sua forma geral não é viável, visto que o mesmo é NP-difícil (GAREY; JOHNSON, 1979). No entanto, há heurísticas eficientes para versões restritas do problema, tal como a especificada para o modelo de tarefas assumido neste capítulo. Uma das mais conhecidas políticas é *Earliest Deadline First* (EDF) (LIU; LAYLAND, 1973).

4.1.1 Política de Escalonamento EDF

As políticas de escalonamento mais utilizadas são baseadas em prioridades. As demais políticas de escalonamento são em geral inflexíveis, possuem um domínio muito específico de aplicação (KOPETZ, 1997) e são em geral incompatíveis requisitos de reconfiguração dinâmica. Sabendo-se que cada tarefa tem uma prioridade associada, de certa forma, facilita a implementação do escalonador, que se limita a escolher, entre as tarefas de Γ prontas para executar, aquela que possui a maior prioridade. Quando as prioridades são fixas em tempo de projeto, diz-se que o escalonador é baseado em prioridade fixa. Por outro lado, políticas de escalonamento baseadas em prioridade variável são aquelas para as quais as prioridades das tarefas são determinadas em tempo de execução. Talvez a representante mais conhecida desta classe seja *Earliest Deadline First* (EDF) (LIU; LAYLAND, 1973). De acordo com esta política, o escalonador escolhe, entre as tarefas prontas para executar, aquela cujo prazo de execução esteja mais próximo a vencer, isto é, aquela que possui o menor *deadline* absoluto.

Algumas políticas de escalonamento baseadas em prioridades variáveis, tal como EDF, são conhecidas como políticas de escalonamento ótimas para o modelo de sistema aqui adotado, ou seja, sistemas uniprocessados compostos de tarefas periódicas ou esporádicas e independentes. Isto significa que caso haja alguma ordem de execução das tarefas segundo a qual nenhum *deadline* é violado para um dado sistema que atende

a este modelo, então não há violação de *deadlines* para a o escalonamento gerado pela política EDF para o mesmo sistema.

Um exemplo simples de escalonamento produzido por EDF para duas tarefas é ilustrado na Figura 4.1. Assume-se nesta ilustração que ambas as tarefas estão prontas para executar a partir do instante inicial. As setas verticais indicam os instantes de ativação das tarefas e os retângulos suas execuções. Note que τ_2 tem maior prioridade que τ_1 no intervalo $[6, 7)$. Nos demais momentos, sua prioridade é menor que τ_1. De fato, no instante 6 o *deadline* absoluto de τ_1 é 9 enquanto que o de τ_1 é 8.

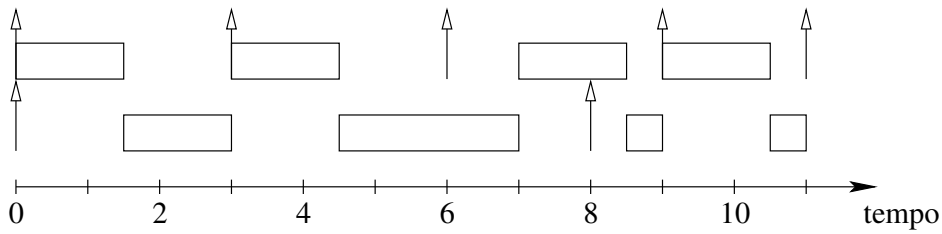


Figura 4.1: Escalonamento EDF para duas tarefas, $\tau_1 = (1.5, 3)$ e $\tau_2 = (4, 8)$

Não basta ter uma política de escalonamento eficiente. É necessário saber que um dado sistema, escalonado pela política escolhida, cumprirá ou não seus *deadlines* quando em execução. Para tanto, lança-se mão de ferramentas de *análise de escalonamento*. É importante ressaltar que métodos baseados em simulação não são satisfatórios devido ao alto número de possibilidades de escalonamento. Diferentemente de simulação, análise de escalonamento baseia-se em funções matemáticas que exprimem o comportamento temporal do sistema no pior caso.

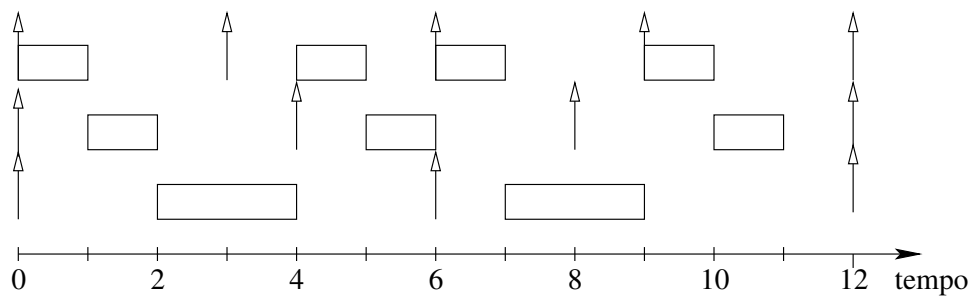
Para o modelo de tarefas assumido neste capítulo, sabe-se que EDF é capaz de produzir um escalonamento correto se e somente se (LIU; LAYLAND, 1973):

$$U = \sum_{\forall \tau_i \in \Gamma} \frac{C_i}{T_i} \leq 1 \tag{4.1}$$

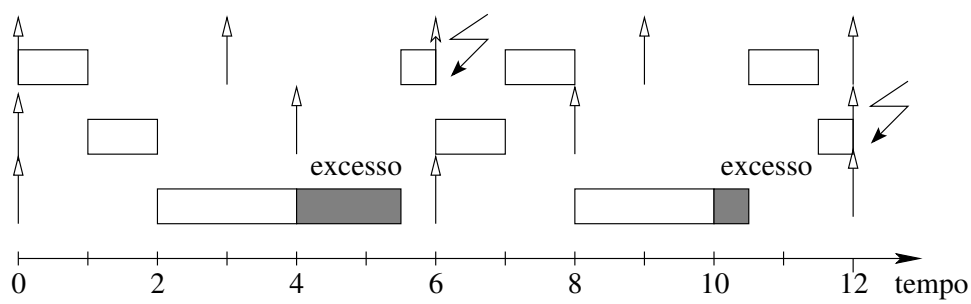
Em outras palavras, como cada tarefa τ_i é ativada no sistema a uma taxa máxima de $1/T_i$ e cada ativação executa no máximo C_i unidades de tempo, a condição acima estabelece que a política EDF pode ser seguramente aplicada a sistemas que não demandam mais que 100% de processador. Este é o caso, por exemplo, da ilustração apresentada na Figura 4.1. Note que esta condição de *escalonabilidade* implica que EDF é uma política ótima, pois, obviamente, não é possível escalonar um sistema que requer mais que 100% de processador. Vale a pena ressaltar que esta otimalidade de

EDF está relacionada apenas a versões restritas do problema. Como no caso geral o problema de escalonamento é NP-difícil, considerando que $\mathcal{P} \neq \mathcal{NP}$, a solução ótima não pode ser gerada por EDF, um algoritmo polinomial.

Um aspecto relevante a ser observado é que para sistemas modernos, com grande variabilidade de custo de execução das tarefas, é muito difícil, ou certamente, impossível, fazer estimativas precisas sobre os valores de C_i. Isto implica que, para tais sistemas, pode haver sobrecargas momentâneas no sistema devido ao fato de o tempo de processamento de algumas tarefas ultrapassar os custos de execução para elas estimados. Como ilustração, considere a Figura 4.2, onde observa-se que um excesso de execução da tarefa τ_3 pode causar perdas de *deadlines* das demais tarefas no sistema.



(a) Não há perdas de *deadlines*.



(b) Perdas de *deadlines* causadas pelo excesso de execução de τ_3

Figura 4.2: Escalonamento EDF para três tarefas, $\tau_1 = (1, 3)$, $\tau_2 = (1, 4)$ e $\tau_3 = (2, 6)$ com possíveis sobrecargas causadas pelo excesso de execução de τ_3, gerando perdas de *deadlines* de maneira arbitrária

Lidar com tais cenários de sobrecarga exige equipar o sistema com mecanismos de proteção. Por exemplo, pode-se usar critérios de descartes de tarefas de tal forma a assegurar que todas as tarefas executadas cumprem seus *deadlines*. As demais, que causem risco de perdas de *deadlines*, são descartadas. Esta abordagem é geralmente

conhecida como *controle de admissão* (KOREN; SHASHA, 1995; RAMANATHAN; HAMDAOUI, 1995; BERNAT; BURNS; LLAMOSI, 2001). Outra alternativa seria empregar mecanismos de *isolamento temporal* entre as tarefas. Isto é geralmente feito através de mecanismos para *reserva de banda de processamento* (MERCER; SAVAGE; TOKUDA, 1993; RAJKUMAR et al., 1998), que provêem o confinamento das possíveis sobrecargas às tarefas com excesso de execução, isolando o efeito de possíveis sobrecargas na execução das demais. Como os mecanismos de reconfiguração dinâmica descritos neste capítulo fazem uso de tais mecanismos, estes serão descritos em maiores detalhes a seguir.

4.1.2 Escalonamento com Isolamento Temporal

Para isolar temporalmente os efeitos da execução de uma tarefa sobre as outras geralmente usa-se o conceito de *servidor* para efetuar reservas de banda de processamento. Um servidor é uma tarefa virtual responsável por escalonar uma ou mais tarefas da aplicação. Cada servidor S_i é representado por uma tupla $S_i = (Q_i, T_i)$. Os parâmetros Q_i e T_i definem a capacidade de S_i e seu período, respectivamente. Tais parâmetros significam que S_i disponibiliza Q_i unidades de tempo a cada período T_i para a execução das tarefas a ele associadas. Por sua vez, servidores podem ser escalonados pelo sistema como se fossem tarefas periódicas.

O comportamento dos servidores está ainda associado à maneira como sua capacidade é gerenciada ao longo da execução do sistema. Para os propósitos deste capítulo, é suficiente escolher um determinado conjunto de regras, apesar de os mecanismos de reconfiguração dinâmica aqui descritos serem relativamente independentes de tais regras. Um servidor amplamente conhecido para sistemas escalonados por EDF é o *Constant Bandwidth Server* (CBS) (ABENI; BUTTAZZO, 2004). Embora outros esquemas de reserva de banda possam ser usados, escolheu-se CBS por se um esquema de escalonamento baseado em reserva de banda amplamente conhecido que é base para várias outras abordagens de escalonamento (CACCAMO; BUTTAZZO; THOMAS, 2005; CACCAMO; BUTTAZZO; SHA, 2000; LIPARI; BARUAH, 2000).

De acordo com suas regras, se $S_i = (Q_i, T_i)$ é um servidor CBS, ele provê uma banda de processamento constante e igual a Q_i/T_i. Esta taxa refere-se à reserva de processamento destinada à execução de S_i, ou seja, destinada à tarefa (ou tarefas) associada(s) a S_i. Todo servidor S_i mantém ainda um atributo, q_i, que representa sua carga ao longo do tempo. Inicialmente, $q_i = Q_i$ e seu valor é decrementado sempre que S_i executa, ou seja, quando alguma tarefa associada a S_i executa. A taxa de decremento de q_i é igual a uma unidade de tempo por unidade de execução de S_i.

Além disto, quando o valor de q_i é nulo, ele é restaurado para Q_i e o *deadline* de S_i é incrementado em T_i.

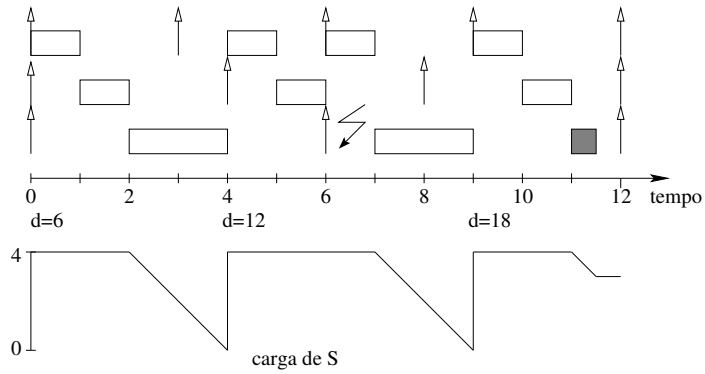


Figura 4.3: Escalonamento EDF para três tarefas, $\tau_1 = (1, 3)$, $\tau_2 = (1, 4)$ e $\tau_3 = (2, 6)$. A primeira e a segunda instâncias de τ_3 apresentam excessos de execução de 2 e 1 unidades de tempo, respectivamente. A execução de τ_3 é gerenciada por um CBS $S = (2, 6)$. Perdas de *deadlines* ficam confinadas à tarefa que apresentou excesso de execução. Folgas de processamento são usadas para executar possíveis excessos, reduzindo possíveis efeitos de sobrecarga

A Figura 4.3 ilustra o efeito de se usar CBS para acomodar a execução da tarefa τ_3 do exemplo apresentado na Figura 4.2, onde τ_3 é associada ao servidor $S = (2, 6)$. Além do escalonamento gerado, a figura informa ainda a evolução da carga de S e seus *deadlines* absolutos. Note que os parâmetros do servidor são iguais às estimativas dos atributos temporais originalmente previstos para τ_3. Como pode ser observado no escalonamento gerado, no intervalo de tempo $[0, 6)$ não há folga de processamento. Neste caso, a perda de *deadline* de τ_3 é observada. Em outras palavras, neste intervalo apenas o percentual de processador que foi previamente garantido e reservado a S é usado e o excesso de execução não pode ser acomodado. Por outro lado, o excesso de execução da segunda ativação de τ_3 pode ser atendida, pois há duas unidades de folga de processamento no intervalo $[8, 10)$, uma delas usada para executar o excesso de τ_3. Outro aspecto importante para ser ressaltado é que as falhas temporais estão confinadas à execução de τ_3 e não mais se propagam para as demais tarefas. Observe que no instante 9, o *deadline* absoluto de S é 18, fazendo com que S tenha a mais baixa prioridade no sistema no intervalo $[9, 12)$. Esta propriedade, de isolamento temporal, é de extrema importância para sistemas que possuem requisitos adaptativos, como os que são alvo dos mecanismos de reconfiguração dinâmica descritos neste capítulo.

4.1.3 Modelo de Reconfiguração Dinâmica

Considera-se um sistema monoprocessado. As tarefas do sistema são escalonadas segundo um esquema baseado em reserva de banda do processador. Em particular, assume-se que existem n servidores do tipo CBS caracterizados pelo conjunto $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$.

Assume-se que cada tarefa é designada a um servidor único S_i de forma que seus parâmetros Q_i e T_i se tornam relacionados ao tempo de execução C_i e intervalo de liberação das instâncias T_i da tarefa correspondente, respectivamente. No entanto, diferentemente de outros trabalhos voltados ao escalonamento baseado em reserva de banda, assume-se que os parâmetros (Q_i, T_i) são determinados dinamicamente, o que permite que sejam redefinidos em tempo de execução por meio de mecanismos de reconfiguração das tarefas do sistema. Além disso, é preferível designar uma tarefa por servidor para que a sobrecarga de uma tarefa não interfira nas outras tarefas do sistema.

Assume-se ainda que não há relações de precedência e compartilhamento de recursos entre as tarefas. Portanto, uma vez que cada servidor é escalonado segundo a política EDF, 100% de utilização do processador pode ser alcançada a fim de preservar a correção do escalonamento dos servidores em $\mathcal{S}$. Ou seja, o sistema de servidores é escalonável se e somente se

$$\sum_{S_i \in \mathcal{S}} \frac{Q_i}{T_i} \leq 1 \tag{4.2}$$

Esta condição é equivalente à condição (4.1) anteriormente descrita para um sistema de n tarefas. Assume-se, portanto, que EDF é utilizado para escalonar o conjunto de servidores e, desta forma, 100% do processador pode ser utilizada.

A garantia de um servidor não perder *deadline* não necessariamente implica na correção temporal da tarefa a ele associada, como exemplificado na Figura 4.3. O mecanismo de reconfiguração objetiva justamente distribuir reservas de processamento entre os servidores de tal forma a minimizar os efeitos colaterais devido às possíveis perdas de *deadlines* de tarefas. Por exemplo, num determinado *modo de operação* pode ser desejável fornecer mais processador para alguns servidores que executam tarefas mais importantes que para outras. Escolher qual a melhor distribuição de banda de processamento, no entanto, geralmente envolve um problema computacionalmente complexo.

Neste contexto, assume-se aqui que o sistema/usuário pode, a qualquer instante, requisitar a reconfiguração dos parâmetros dos servidores em $\mathcal{S}$. Tal requisição pressupõe que haja conhecimento prévio sobre qual a utilização $U_i = Q_i/T_i$ ideal para

executar cada servidor $S_i = (Q_i, T_i)$ e o benefício v_i relativo de cada servidor S_i para o sistema. Por exemplo, num sistema de monitoramento, o usuário pode escolher ampliar ou melhorar os detalhes da imagem vinda de determinadas câmeras. Ao efetuar tal escolha, o operador estaria informando que determinado servidor S_i passa a ter benefício v_i e, para executá-lo, precisa-se de U_i. O mecanismo de reconfiguração então redistribui os recursos de processamento a fim de maximizar o valor de benefício agregado de todos os servidores. O valor de benefício efetivamente atingido irá depender da largura de banda u_i que o sistema for capaz de reservar. Se o sistema tem capacidade para alocar apenas $u_i < U_i$, o benefício efetivo será inferior a v_i. Além disso, não há benefício adicional ao ser alocar $u_i > U_i$ o que deve ser levado em consideração pelo procedimento de reconfiguração.

Como pode ser notado, a reconfiguração de parâmetros (Q_i, T_i) de servidores demanda a resolução de um problema de otimização, tendo a equação (4.2) como uma das restrições. A função objetivo do problema será descrita nas próximas seções, já que, como será visto, esta função dependerá da versão do problema de otimização a ser empregado. Primeiro, assume-se que os tempos de execução das tarefas são determinísticos (*e.g.*, tempo de pior caso). Esta hipótese é adequada quando exige-se garantias de escalonabilidade determinísticas. Mais tarde, esta hipótese será relaxada e os tempos de execução se tornarão variáveis aleatórias. Neste caso, será possível maximizar o benefício total induzido pelo sistema enquanto se assegura escalonabilidade com um probabilidade definida pelo usuário/projetista.

É importante mencionar que quando da reconfiguração do sistema será necessário realizar a mudança de um modo de operação de certas tarefas para outros modos. Durante o período em que o sistema está mudando os modos de operação, poderão coexistir tarefas executando em diferentes modos de operação, o que poderá causar perdas de *deadlines* durante a transição. Para assegurar que *deadlines* não sejam perdidos em sistemas sob escalonamento EDF com diferentes modos de operação, é suficiente que o sistema não utilize mais do que 50% do processador (ANDERSSON, 2008). Portanto, se for necessário evitar a possibilidade de perda de *deadlines* antes, após e durante a troca de modos, uma perda de 50% da capacidade de processamento será necessária. Para tanto, é suficiente a redução do limite superior aplicado na equação (4.2) para $1/2$. Outras estratégias para evitar perda de *deadlines* também podem ser consideradas. Por exemplo, os servidores que exigem mais capacidade de processamento no novo modo de operação poderiam aguardar até que seus *deadlines* sejam atingidas para então efetivar a mudança de modo de operação (BUTTAZZO; ABENI, 2002).

No entanto, o interesse aqui está em sistemas que permitem falhas transientes (*soft real-time systems*) e, portanto, uma estratégia de mudança de modo de operação me-

nos restritiva poderia ser empregada. Por exemplo, antes de aplicar o novo modo de reconfiguração, as tarefas que estão executando no modo anterior poderiam ser canceladas. Alternativamente, poderia-se permitir a execução até que sejam finalizadas sob risco de perderem seus *deadlines*. Uma vez que isolamento temporal é assegurado pelo mecanismo CBS, falhas temporais transientes em algumas tarefas não se propagam a outras tarefas. Neste capítulo assume-se que estratégias simples como estas poderiam ser aplicadas.

4.2 Reconfiguração com Parâmetros Determinísticos

Esta seção se concentra em modelos e algoritmos para reconfiguração de servidores em situações onde a utilização dos servidores é um valor determinístico. O primeiro modelo assume que a utilização u_i do servidor S_i pode ser configurada em um valor de um conjunto finito $\{u_{i,1}, \ldots, u_{i,\kappa(i)}\}$, enquanto o segundo modelo assume que a utilização pode ser ajustada continuamente dentro do intervalo $[L_i, U_i]$.

4.2.1 Modelo Discreto

Para cada servidor S_i, o modelo assume que existem $\kappa(i)$ valores para u_i. Com $K_i = \{1, 2, \ldots, \kappa(i)\}$ e $u_{i,k} = Q_{i,k}/T_{i,k}$, $k \in K_i$, define-se a k-ésima configuração de S_i para cada uma das quais o benefício é dado por $A_{i,k}$. Com estas definições, o problema de otimização a ser resolvido pelo mecanismo de reconfiguração dinâmica é formulado como segue:

$$PD : fd = \max \sum_{(i,k)\in\Omega} A_{i,k} x_{i,k} \tag{4.3a}$$

$$\text{s.a} : \sum_{(i,k)\in\Omega} u_{i,k} x_{i,k} \leq 1 \tag{4.3b}$$

$$\sum_{k\in K_i} x_{i,k} = 1,\ S_i \in \mathcal{S} \tag{4.3c}$$

$$x_{i,k} \in \{0,1\}, (i,k) \in \Omega \tag{4.3d}$$

onde $\Omega = \{(i,k) : S_i \in \mathcal{S},\ k \in K_i\}$. As variáveis de decisão $x_{i,k}$ determinam a configuração $k \in K_i$ escolhida para cada servidor S_i. A restrição (4.3b) garante a escalonabilidade de $\mathcal{S}$ conforme a política EDF. A restrição (4.3c) garante que exatamente uma configuração é selecionada. Sem perda de generalidade, assume-se que

$u_{i,k} \geq u_{i,k-1}$ para todo $S_i \in \mathcal{S}$ e $k \geq 2$. A possibilidade de interromper a execução de um servidor S_i pode ser implementada definindo-se $A_{i,1} = 0$ e $u_{i,1} = 0$. Para se admitir um novo servidor S_i no sistema em tempo de execução, é suficiente definir $A_{i,k} > 0$ e $u_{i,k} > 0$, $k = 1, \ldots, \kappa(i)$. Assume-se ainda que $\sum_{S_i \in \mathcal{S}} u_{i,1} < 1$ pois de outra forma os servidores não seriam escalonáveis ou não se teria potencial para otimizar as aplicações. O parâmetro $A_{i,k}$ define o benefício induzido ao se alocar $u_{i,k}$ unidades de utilização do processador ao servidor S_i, conforme a equação:

$$A_{i,k} = \frac{\min(u_{i,k}, u_i)}{u_i} v_i \,, \tag{4.4}$$

A função objetivo (4.4) pode ser interpretada como segue. Se o sistema é capaz de alocar pelo menos u_i a S_i, um valor de benefício $v_i \geq 0$ é atingido. O valor v_i deve ser derivado a partir do conhecimento do projetista do sistema, constituindo um dos parâmetros utilizados pelo mecanismo de reconfiguração. O valor v_i não é assumido estático, podendo variar com o decorrer do tempo ou em resposta a eventos do sistema. O sistema/usuário, a qualquer instante, pode mudar estes valores.

Exemplo Ilustrativo

A instância ilustrativa do problema de reconfiguração de servidores compreende $n = 4$ servidores. A Tabela 4.1 apresenta a utilização de CPU para $\kappa(i) = 5$ configurações de cada servidor $S_i \in \mathcal{S}$. O benefício de cada configuração é mostrado na Tabela 4.2.

Tabela 4.1: Utilização de CPU das configurações dos servidores

	Servidor S_i			
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$u_{i,1}$	10/100	30/200	10/200	7/100
$u_{i,2}$	15/50	25/100	08/40	9/60
$u_{i,3}$	50/100	16/40	21/60	10/50
$u_{i,4}$	35/50	110/200	30/50	15/50
$u_{i,5}$	56/70	30/40	65/100	40/100

Fundamentos

O problema clássico da mochila é trivialmente reduzido ao problema de reconfiguração de servidores, logo PD é um problema NP-Difícil (WOLSEY, 1998). Não surpreendentemente, algoritmos de programação dinâmica podem ser facilmente adaptados para o problema PD. Seja $PD(i, \lambda)$ a versão de PD restrita aos servidores

Tabela 4.2: Benefícios das configurações dos servidores

	Servidor S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
$A_{i,1}$	0,125	0,200	0,076	0,175
$A_{i,2}$	0,375	0,333	0,307	0,375
$A_{i,3}$	0,625	0,533	0,538	0,500
$A_{i,4}$	0,875	0,733	0,923	0,750
$A_{i,5}$	1,000	1,000	1,000	1,000

$\mathcal{S}_i = \{S_1, \ldots, S_i\}$ e com uma disponibilidade de CPU $\lambda \leq 1$. O algoritmo de programação dinâmica padrão (primal) assume que os parâmetros $u_{i,k}$ são todos inteiros, o que pode ser induzido ao se multiplicar ambos os lados da restrição (4.3b) por um inteiro Λ suficientemente grande. A ideia por trás do algoritmo de programação dinâmica está na construção gradual de uma solução resolvendo-se problemas cada vez mais complexos. Mais precisamente, resolve-se uma família $\{PD(i,\lambda)\}$ de problemas usando a recursão:

$$PD(i,\lambda) : fd(i,\lambda) = \max \sum_{k \in K_i} A_{i,k} x_{i,k} + fd(i-1, \lambda - \lambda_i) \tag{4.5a}$$

$$\text{s.a} : \lambda_i = \sum_{k \in K_i} \Lambda u_{i,k} x_{i,k} \tag{4.5b}$$

$$\sum_{k \in K_i} x_{i,k} = 1 \tag{4.5c}$$

$$x_{i,k} \in \{0,1\},\ k \in K_i \tag{4.5d}$$

onde $fd(i,\lambda) = -\infty$ se $\lambda < \Lambda u_{i,1}$. Em outras palavras, $fd(i,\lambda)$ é o benefício do sistema induzido pelo escalonamento dos servidores $\mathcal{S}_i$ usando apenas λ unidades de recurso. O caso terminal ocorre quando $i = 0$, para o qual $fd(0,\lambda) = 0$ se $\lambda \geq 0$ e $fd(0,\lambda) = -\infty$ se $\lambda < 0$.

A computação de $fd(i,\lambda)$ envolve a configuração do servidor S_i em um dos modos disponíveis, digamos no modo k. Então, os servidores restantes $S_1, S_2, \ldots, S_{i-1}$ deverão ser configurados usando apenas a quantidade de recursos restante, $\lambda - \Lambda u_{i,k}$, e o benefício total de tal configuração será $fd(i,\lambda) = A_{i,k} + fd(i-1, \lambda - \Lambda u_{i,k})$, que por sua vez é computado recursivamente.

A versão dual do algoritmo de programação dinâmica inverte os papéis de objetivo e restrição, portanto minimiza a quantidade de recursos necessária para se atingir um dado benefício. Ilustrações da aplicação do algoritmo de programação dinâmica primal e dual à instância exemplo são encontrados em (OLIVEIRA, 2009).

Obviamente, $PD \equiv PD(n, \Lambda)$ e portanto $fd = fd(n, \Lambda)$. A formulação recursiva (4.5a)–(4.5d) leva imediatamente a um algoritmo de programação dinâmica que executa em tempo $O(\Lambda n \max\{\kappa(i) : S_i \in \mathcal{S}\})$, portanto pseudo-polinomial. Contudo, o tempo de execução pseudo-polinomial pode ser demasiado lento para aplicações tempo-real, estimulando o desenvolvimento de algoritmos de aproximação.

Um algoritmo que produz uma solução aproximada do ótimo é dito algoritmo de aproximação (MOTWANI, 1992). Tais algoritmos são úteis na busca de uma boa solução quando o tempo computacional é restrito, especialmente quando se trata de um problema NP-Difícil. Abaixo serão apresentados conceito úteis para o projeto de dois algoritmos de aproximação, ambos baseados em estratégias gulosas.

Uma instância $I = (u_{i,k}, A_{i,k} : (i,k) \in \Omega)$ consiste dos parâmetros que definem um problema PD específico. Um vetor $\mathbf{x} = (x_{i,k} : (i,k) \in \Omega)$ é factível se este satisfaz as restrições (4.3b) a (4.3d). $S(I) = \{\mathbf{x} : \mathbf{x}$ satisfaz as restrições (4.3b) a (4.3d)$\}$ é o conjunto de soluções factíveis. Uma solução factível $\mathbf{x}^\star$ é ótima se $fd(\mathbf{x}^\star) \geq fd(\mathbf{x})$ para todo $\mathbf{x} \in S(I)$. $OPT(I) = fd(\mathbf{x}^\star)$ denota o objetivo ótimo.

Definição 1 *Um algoritmo A que produz uma solução factível $\mathbf{x} \in S(I)$ para uma instância I de PD é dito* **algoritmo de aproximação**. *$A(I)$ denota o valor $fd(\mathbf{x})$ da solução produzida por A.*

Definição 2 *Seja A um algoritmo de aproximação para PD. A razão relativa de desempenho $R_A(I)$ de A para uma instância de entrada I é definida como $R_A(I) = \frac{OPT(I)}{A(I)}$.*

Algoritmo de Densidade Guloso

O algoritmo de densidade guloso (ADG) possui dois passos distintos. O primeiro passo aloca o mínimo de recursos necessários para assegurar a factibilidade da reconfiguração. O segundo passo distribui os recursos remanescentes aos servidores seguindo uma ordem não crescente de benefício relativo adicional (densidade),

$$\widehat{A}_{i,k} = \frac{(A_{i,k} - A_{i,1})}{(u_{i,k} - u_{i,1})}$$

como chave primária e em ordem não crescente de demanda de recurso adicional,

$$\widehat{u}_{i,k} = (u_{i,k} - u_{i,1})$$

como chave de ordenação secundária.

Em essência, ADG resolve a relaxação de programação linear (PL) de uma versão modificada de PD e depois usa arredondamento para produzir uma solução factível. Para assegurar o mínimo de recursos a todos os servidores, obtém-se uma versão modificada PD' na qual o benefício de cada modo do servidor S_i é $A'_{i,k} = (A_{i,k} - A_{i,1})$ com um consumo de recurso $\widehat{u}_{i,k}$ para todo $k > 1$. Se as variáveis binárias $x_{i,k}$, $S_i \in \mathcal{S}$ e $k \in K_i - \{1\}$, são relaxadas para $x_{i,k} \in [0,1]$, então uma solução ótima para PD' é facilmente obtida distribuindo os recursos remanescentes aos servidores segundo uma ordem não crescente de densidade. Ou seja, os servidores são configurados em níveis diferentes dos níveis de consumo mínimo, com prioridade dada àqueles com máximo benefício adicional por unidade de recurso adicional consumida, que é precisamente $\widehat{A}_{i,k}$ (densidade). Estas ideias são formalizadas no **Algoritmo** 4.1. Dado qualquer subconjunto de servidores e níveis de reconfiguração $\omega \subseteq \Omega$, seja $S(\omega) = \{S_i : (i,k) \in \omega\}$ o conjunto de servidores presentes em ω.

Algorithm 4.1 Algoritmo de densidade guloso para reconfiguração discreta de servidores

input: Servers $\mathcal{S}$, benefit values $A_{i,k}$, and utilizations $u_{i,k}$
sorting: Order the pairs of $\Omega - \{(i,1) : S_i \in \mathcal{S}\}$ in the sequence $\langle (i_1,k_1), \ldots, (i_{|\Omega|-|\mathcal{S}|}, k_{|\Omega|-|\mathcal{S}|})\rangle$ so that $\widehat{A}_{i_p,k_p} > \widehat{A}_{i_q,k_q}$ or $\widehat{A}_{i_p,k_p} = \widehat{A}_{i_q,k_q}$ and $\widehat{u}_{i_p,k_p} \geq \widehat{u}_{i_q,k_q}$ for every $p < q$
$\omega := \emptyset$
$b := 1 - \sum_{S_i \in \mathcal{S}} u_{i,1}$
$t := 1$
while $t \leq (|\Omega| - |\mathcal{S}|)$ and $|\omega| < |\mathcal{S}|$ **do**
    if $S_{i_t} \notin S(\omega)$ and $\widehat{u}_{i_t,k_t} \leq b$ **then**
        $\omega := \omega \cup \{(i_t, k_t)\}$
        $b := b - \widehat{u}_{i_t,k_t}$
    end if
    $t := t + 1$
end while
for all $S_i \in \mathcal{S} - S(\omega)$ **do**
    $\omega := \omega \cup \{(i,1)\}$
end for
return ω

O algoritmo de densidade guloso executa em tempo $O(|\Omega| \lg |\Omega|)$, que é em essência o custo computacional necessário para ordenar os elementos de Ω com tamanho limitado por $O(n \max\{\kappa(i) : S_i \in \mathcal{S}\})$.

Um algoritmo de aproximação modificado chamado *algoritmo de densidade guloso modificado* ($ADGM$) pode ser obtido a partir de ADG. $ADGM$ retorna a configuração $\omega \subseteq \Omega$ produzida por ADG, a menos que a reconfiguração de um servidor $S_{i'}$ em um modo de operação k', (i',k'), induza um valor objetivo maior que o objetivo produzido por ADG. Este algoritmo está formalizado no **Algoritmo** 4.2. Neste último caso, $ADGM$ retorna $\omega' = \{(i,1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i',k')\}$. Seja

$\mathbf{x}(\omega) = (x_{i,k} : (i,k) \in \Omega)$ um vetor tal que $x_{i,k} = 1$ para todo $(i,k) \in \omega$ enquanto $x_{i,k} = 0$ para todo $(i,k) \in \Omega - \omega$.

Algorithm 4.2 Algoritmo de densidade guloso modificado para reconfiguração discreta de servidores

1: **input:** Servers $\mathcal{S}$, benefit values $A_{i,k}$, and utilizations $u_{i,k}$
2: $(i',k') := \arg\max_{(i,k)\in\Omega:k>1} \{A_{i,k} - A_{i,1} : \widehat{u}_{i,k} \leq 1 - \sum_{S_t \in \mathcal{S}} u_{t,1}\}$
3: $\omega' := \{(i,1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i',k')\}$
4: $\omega := ADG(\mathcal{S}, \{A\}, \{u\})$
5: **if** $fd(\mathbf{x}(\omega')) > fd(\mathbf{x}(\omega))$ **then**
6: **return** ω'
7: **else**
8: **return** ω
9: **end if**

Teorema 1 *Para qualquer instância I de PD, o desempenho do algoritmo de densidade guloso modificado, denotado por $ADGM(I)$, e o desempenho ótimo, denotado por $OPT(I)$, são relacionados pela seguinte expressão:*

$$ADGM(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i,1}}{2} \tag{4.6}$$

Prova Seja $\omega^\star$ a solução produzida por $ADGM$, onde $\omega^\star = \omega$ se $fd(\omega) \geq fd(\omega')$ e $\omega^\star = \omega'$ caso contrário. Seja $\hat{\omega}$ a solução obtida seguindo ADG até a primeira iteração t na qual ADG falha ao tentar trocar o nível do servidor S_{i_t} de 1 para k_t.

Claramente, (i_t, k_t) é igual a $(i_{\hat{\omega}}, k_{\hat{\omega}}) = \arg\max_{(i,k)\in\Omega:k>1} \{(A_{i,k} - A_{i,1})/(u_{i,k} - u_{i,1}) : (i,1) \in \hat{\omega}\}$.

Claramente, $OPT(I)$ não pode ser maior que o valor da solução ótima para a relaxação de programação linear, portanto

$$\begin{aligned} OPT(I) &\leq fd(\hat{\omega}) + (A_{i_{\hat{\omega}},k_{\hat{\omega}}} - A_{i_{\hat{\omega}},1}) \frac{1 - \sum_{(i,k)\in\hat{\omega}} u_{i,k}}{u_{i_{\hat{\omega}},k_{\hat{\omega}}} - u_{i_{\hat{\omega}},1}} \\ &\leq fd(\omega) + (A_{i_{\hat{\omega}},k_{\hat{\omega}}} - A_{i_{\hat{\omega}},1}) \\ &= fd(\omega) + (A_{i_{\hat{\omega}},k_{\hat{\omega}}} + \sum_{S_i \in \mathcal{S}: i \neq i_{\hat{\omega}}} A_{i,1}) - \sum_{S_i \in \mathcal{S}} A_{i,1} \end{aligned}$$

onde a segunda desigualdade segue do fato que $fd(\omega) \geq fd(\hat{\omega})$ e $\frac{1-\sum_{(i,k)\in\hat{\omega}} u_{i,k}}{u_{i_{\hat{\omega}},k_{\hat{\omega}}} - u_{i_{\hat{\omega}},1}} < 1$.

Há dois casos. Se $\omega^\star = \omega$, então $fd(\omega) \geq fd(\omega') \geq (A_{i_{\hat{\omega}},k_{\hat{\omega}}} + \sum_{S_i \in \mathcal{S}: i \neq i_{\hat{\omega}}} A_{i,1})$ e portanto:

$$OPT(I) \leq 2fd(\omega) - \sum_{S_i \in \mathcal{S}} A_{i,1} = 2ADGM(I) - \sum_{S_i \in \mathcal{S}} A_{i,1}$$
$$\implies ADGM(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i,1}}{2}$$

De outra forma, se $\omega^\star = \omega'$, então $fd(\omega') \geq (A_{i_{\hat{\omega}},k_{\hat{\omega}}} + \sum_{S_i \in \mathcal{S}: i \neq i_{\hat{\omega}}} A_{i,1})$ e portanto:

$$OPT(I) \leq 2fd(\omega') - \sum_{S_i \in \mathcal{S}} A_{i,1} = 2ADGM(I) - \sum_{S_i \in \mathcal{S}} A_{i,1}$$
$$\implies ADGM(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i,1}}{2}$$

Portanto, demonstra-se a relação (4.6) entre $OPT(I)$ e $ADGM(I)$. ∎

Corolário 1 *O desempenho relativo do algoritmo de densidade guloso modificado é* $R_{ADGM} \leq 2$.

Prova A partir do teorema acima, tem-se que:

$$\frac{OPT(I)}{ADGM(I)} \leq \frac{OPT(I)}{(OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i,1})/2} \leq 2 \tag{4.7}$$

Demonstrando que $R_{ADGM} \leq 2$. ∎

Um esquema de aproximação completamente polinomial (*Fully-Polynomial Approximation Scheme*, FPAS) do problema clássico da mochila foi estendido para resolver o problema PD (OLIVEIRA, 2009). Este FPAS toma como base o algoritmo de programação dinâmica dual o qual troca os papéis de objetivo e recurso. O algoritmo $ADGM$ produziu soluções quase-ótimas com pouco esforço quando comparado ao FPAS.

A Tabela 4.3 mostra os valores $\hat{A}_{i,k}$, e as respectivas utilizações $u_{i,k}$, obtidos a partir do exemplo ilustrativo apresentado nas Tabelas 4.1 e 4.2. A execução de ADG sobre a instância exemplo produz o seguinte resultado. O valor inicial de b é $0,63$. A primeira interação do **while** seleciona $\omega = \{(4,5)\}$, definindo $b = 0,3$. Após a segunda iteração, $\omega = \{(4,5),\ (3,3)\}$ e $b = 0$, o que significa que nenhuma melhoria adicional é possível. A reconfiguração final é $\omega = \{(4,5),\ (3,3),\ (1,1),\ (2,1)\}$ que induz um valor de benefício de $fd(\mathbf{x}(\omega)) = 1,863$. $ADGM$ não consegue melhorar este

resultado. De fato, $(i', k') = (1, 5)$ que resulta em $\omega' = \{(1, 5), (2, 1), (3, 1), (4, 1)\}$ e induz o benefício $fd(\mathbf{x}(\omega')) = 1,451$. De fato, $fd(\mathbf{x}(\omega)) = 1,863$ é o benefício máximo para a instância exemplo.

Tabela 4.3: Benefício relativo dos modos operacionais dos servidores

	$(\hat{A}_{i,k}, u_{i,k})$			
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$k = 1$	$(0,00;\ 0,10)$	$(0,00;\ 0,15)$	$(0,00;\ 0,05)$	$(0,00;\ 0,07)$
$k = 2$	$(1,25;\ 0,30)$	$(1,33;\ 0,25)$	$(1,54;\ 0,20)$	$(2,50;\ 0,15)$
$k = 3$	$(1,25;\ 0,50)$	$(1,33;\ 0,40)$	$(1,54;\ 0,35)$	$(2,50;\ 0,20)$
$k = 4$	$(1,25;\ 0,70)$	$(1,33;\ 0,55)$	$(1,54;\ 0,60)$	$(2,50;\ 0,30)$
$k = 5$	$(1,25;\ 0,80)$	$(1,33;\ 0,75)$	$(1,54;\ 0,65)$	$(2,50;\ 0,40)$

Avaliação de Desempenho do Algoritmo de Aproximação

Problemas com cardinalidade e número de modos operacionais variados foram gerados para avaliar o desempenho dos métodos de aproximação, especificamente as instâncias variaram segundo $25 \leq |\mathcal{S}| \leq 250$ e $3 \leq \kappa(i) \leq 10$.

Obviamente, um sistema com 250 tarefas é improvável de ser implementado mas foram considerados com o objetivo de avaliar o desempenho dos algoritmos. As configurações foram geradas usando uma distribuição uniforme para utilizações e valor de benefício. O tempo de execução médio sobre 5 instâncias de mesmo tamanho foi considerado para minimizar erros de medição do tempo computacional.

A Figura 4.4 mostra os tempos de execução de ADG e $ADGM$. Uma comparação com o algoritmo de programação dinâmica (não apresentada aqui) mostra que estes métodos são cerca de vinte mil vezes mais rápidos.

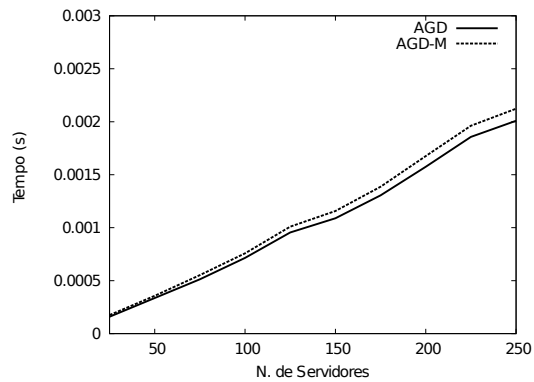


Figura 4.4: Tempo de execução de ADG e $ADGM$

A Figura 4.5 mostra a qualidade da solução retornada pelos esquemas de aproximação em termos absolutos. A solução ótima apresentada na Figura 4.5 foi obtida por meio do algoritmo de programação dinâmica, enquanto que o limite inferior para a aproximação e o limite superior para a solução ótima foram calculados utilizando-se a desigualdade (4.6) do Teorema 1. Pode-se observar que as soluções retornadas por $ADGM$ são muito mais próximas do ótimo do que o limite inferior teórico, induzido pela desigualdade (4.7).

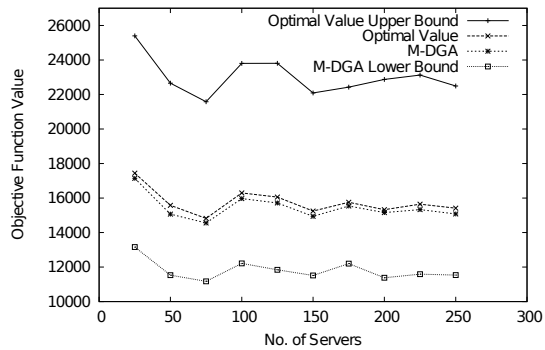


Figura 4.5: Desempenho absoluto dos algoritmos de aproximação

4.2.2 Modelo Contínuo

O modelo contínuo é caracterizado pela ausência de configurações pré-definidas dos servidores S_i de $\mathcal{S}$. Logo, esta forma de reconfiguração consiste em encontrar valores para x_i, $L_i \leq x_i \leq U_i$, que maximizam o benefício global do sistema, onde L_i e U_i são o limite inferior e superior para x_i, respectivamente[1]. Mais formalmente, o problema de reconfiguração contínua é da forma:

$$PC : fc = \max \sum_{S_i \in \mathcal{S}} A_i x_i \tag{4.8a}$$

$$\text{s.a} : \sum_{S_i \in \mathcal{S}} x_i \leq 1 \tag{4.8b}$$

$$L_i \leq x_i \leq U_i,\ S_i \in \mathcal{S} \tag{4.8c}$$

[1] A utilização do servidor S_i é definida como x_i em vez de u_i. Desta forma, as variáveis de decisão estão consistentes com o modelo discreto; simplifica-se ainda a notação para utilizações estocásticas (Seção 4.3).

Nesta função objetivo, o benefício A_i é acumulado proporcionalmente à largura de banda alocada ao servidor S_i. Semelhante à formulação discreta, a escalonabilidade do sistema é assegurada com a equação (4.8b). Além disso, nenhum benefício adicional é obtido alocando-se $x_i > U_i$, o que é evitado com a equação (4.8c).

Exemplo Ilustrativo

O exemplo ilustrativo toma a forma dada na Tabela 4.4. Os valores mínimo e máximo de largura de banda para cada servidor $S_i \in \mathcal{S}$, L_i e U_i, são tomados a partir dos valores Q_i/T_i apresentados na Tabela 4.1. Como pode ser observado, os valores de benefício são dados por servidor (ou aplicação) e representam a importância relativa de um servidor no sistema. Os valores de A_i apresentados na Tabela 4.4 são baseados nos valores da Tabela 4.2 mas normalizados por unidade de utilização do processador.

Tabela 4.4: Valores de benefício e limites para largura de banda dos servidores

	Servidor S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
L_i	0,10	0,15	0,05	0,07
U_i	0,80	0,75	0,65	0,40
A_i	1,25	1,33	1,54	2,50

Uma Solução Analítica

Seja $L = \sum_{i=1}^{n} L_i$ o mínimo e $U = \min(1, \sum_{i=1}^{n} U_i)$ o máximo de largura de banda que poderiam ser alocados aos servidores $\mathcal{S}$. A ideia por trás da solução analítica de PC é informalmente explicada como segue. Primeiro, é necessário alocar a banda mínima L_i requerida por cada servidor S_i. A largura de banda residual, $U - L$, é então distribuída aos servidores em $\mathcal{S}$ segundo uma ordem não crescente de taxa de benefício. Seja $S_{(1)}, S_{(2)}, \ldots, S_{(n)}$ uma ordem de $\mathcal{S}$ tal que $A_{(i)} \geq A_{(i+1)}$ para todo $i < n$. Para os servidores em $\mathcal{S}$ apresentados na Tabela 4.4, esta ordem é dada por $S_{(1)} = S_4, S_{(2)} = S_3, S_{(3)} = S_2$ e $S_{(4)} = S_1$. Logo, se $U_{(1)} - L_{(1)} \leq U - L$, então é possível alocar uma largura de banda $U_{(1)}$ a $S_{(1)}$. A alocação da largura de banda restante, *i.e.* $U - (L - L_{(1)} + U_{(1)})$, segue o mesmo procedimento considerando o segundo servidor com maior benefício relativo, $S_{(2)}$, e assim sucessivamente. Seja $S_{(i^\star)}$ o último servidor para o qual é possível alocar banda adicional segundo este procedimento, $U_{(i^\star)}$. Neste caso, a largura de banda do servidor $S_{(i^\star+1)}$ deve ser ajustada ao máximo possível considerando que:

1. a largura de banda $U_{(i)}$ já foi alocada ao servidor $S_{(i)}$, $i = 1, \ldots, i^\star$, e

2. os demais servidores $S_{(j)}$, $j = i^\star + 2, \ldots, n$, receberam o mínimo de largura de banda $L_{(j)}$.

Não é difícil ver que o valor $i^\star$ pode ser calculado como segue:

$$i^\star = \max_{i=0}^{n} \left\{ i : \sum_{j=1}^{i} (U_{(j)} - L_{(j)}) \leq U - \sum_{j=1}^{n} L_{(j)} \right\}$$

Note que se $i^\star = 0$, então não é possível alocar $U_{(1)}$ a $S_{(1)}$. Também, se $i^\star = n$, a largura de banda de todos os servidores $S_{(i)} \in \mathcal{S}$ se torna $U_{(i)}$. A partir da teoria de programação linear, pode-se mostrar que uma solução ótima para PC é:

$$x_{(i)} = \begin{cases} U_{(i)}, & \text{se } i \leq i^\star \\ U - \sum_{j=1}^{i^\star} U_{(j)} - \sum_{j=i^\star+2}^{n} L_{(j)}, & \text{se } i = i^\star + 1 \\ L_{(i)}, & \text{se } i \geq i^\star + 2 \end{cases} \tag{4.9}$$

Claramente, a equação (4.9) implica $\sum_{i=1}^{n} x_i = U$ que é uma condição necessária para a solução $\mathbf{x} = (x_1, \ldots, x_n)$ ser ótima. O lema e o teorema abaixo mostram concisamente a otimalidade da solução derivada desta equação.

Lema 1 *Se $\mathbf{x} = (x_1, \ldots, x_n)$ é uma solução ótima para o problema PC, então $\sum_{i=1}^{n} x_i = U$.*

Prova Seja $\sum_{i=1}^{n} x_i = U' < U$. Então, deve existir x_i e $\epsilon > 0$ tal que $U' + \epsilon \leq U$ e $x_i + \epsilon \leq U_i$. Seja $x'_j = x_j$ para todo $j \neq i$ e $x'_i = x_i + \epsilon$. Uma vez que a função objetivo é monotonicamente crescente em x_i, fica claro que $fc(\mathbf{x}') > fc(\mathbf{x})$. Repetindo esta operação chega-se a condição $\sum_{i=1}^{n} x'_i = U$. ∎

Teorema 2 *O problema PC tem uma solução ótima dada pela equação* (4.9).

Prova A prova é por contradição. Considere um sistema $\mathcal{S}$ composto por n servidores com benefícios dados em ordem não crescente conforme $A_{(1)}, \ldots, A_{(n)}$. Assuma que $\mathbf{x} = (x_1, \ldots, x_n)$ é uma solução derivada da equação (4.9) mas que não seja ótima. Logo, deve existir $\mathbf{x}' = (x'_1, \ldots, x'_n)$ tal que $fc(\mathbf{x}') > fc(\mathbf{x})$. Se $\alpha_i = x'_i - x_i$, então:

$$fc(\mathbf{x}') - fc(\mathbf{x}) = \sum_{x'_{(j)} > x_{(j)}} \alpha_{(j)} A_{(j)} + \sum_{x'_{(k)} < x_{(k)}} \alpha_{(k)} A_{(k)} > 0 \tag{4.10}$$

Se $x'_{(j)} > x_{(j)}$, pode-se concluir $j \geq i^\star + 1$ porque $x_{(j)} = U_{(j)}$ para todo $j < i^\star + 1$. Além disso, se $x'_{(k)} < x_{(k)}$, $k \leq i^\star + 1$ uma vez que $x_{(k)} = L_{(k)}$ para todo $k > i^\star + 1$. A

partir destas observações e do fato que $A_{(i+1)} \leq A_{(i)}$ para todo $i < n$, a desigualdade (4.10) implica que:

$$A_{(i^\star+1)} \sum_{x'_{(j)} > x_{(j)}} \alpha_{(j)} + A_{(i^\star+1)} \sum_{x'_{(k)} < x_{(k)}} \alpha_{(k)} > 0$$

Pela equação (4.9) sabe-se que $\sum_{i=1}^{n} x_i = U$. Ainda, pelo Lema 1, pode-se assumir que $\sum_{i=1}^{n} x'_i = U$. Assim,

$$\alpha = \sum_{x'_{(j)} > x_{(j)}} \alpha_{(j)} = - \sum_{x'_{(k)} < x_{(k)}} \alpha_{(k)}$$

e portanto chega-se a $\alpha(A_{(i^\star+1)} - A_{(i^\star+1)}) > 0$, uma contradição. ∎

Tão logo os valores ótimos de x_i sejam determinados, o próximo passo é a computação dos parâmetros (Q_i, T_i) dos servidores. Isto pode ser realizado fixando-se um dos parâmetros, digamos T_i, tal que $Q_i = x_i T_i$. Por exemplo, se T_i está relacionado a uma taxa de quadro por segundo para uma cena de vídeo, Q_i representaria a largura de banda que induz o benefício global ótimo do sistema.

Na verdade, não há ganho em se permitir que o orçamento Q_i e período T_i se tornem variáveis de decisão, tal que Q_i/T_i venha a substituir x_i com $Q_i^{\min} \leq Q_i \leq Q_i^{\max}$ e $T_i^{\min} \leq T_i \leq T_i^{\max}$ definindo os limites. Seja PC' esta formulação. Pode-se mostrar que PC e PC' são equivalentes (OLIVEIRA, 2009).

Vale observar que a solução analítica dada pela equação (4.9) tem complexidade $O(n)$ assumindo que os valores de benefício estão em ordem não crescente. Este caso representa cenários onde os valores de benefício são designados estaticamente. De outra forma, quando os valores de benefício podem mudar em tempo de execução, uma ordenação dos servidores se faz necessária e o tempo de execução se torna $O(n \lg n)$. O algoritmo não é apresentado por se entender que é uma implementação direta e simples da equação (4.9).

4.3 Reconfiguração com Parâmetros Estocásticos

Primeiramente, realiza-se uma breve introdução a restrições probabilísticas que será um conceito chave para modelar a restrição de escalonabilidade em um contexto probabilístico. Na sequência, os problemas de reconfiguração discreto e contínuo são estendidos para lidar com utilização estocástica. Ao final da seção é apresentado um

método para modelagem de uma variável aleatória arbitrária usando uma variável aleatória Gaussiana que aproxima a distribuição de probabilidade cumulativa.

4.3.1 Restrições Probabilísticas

Seja $\mathbf{u}$ uma variável aleatória com domínio contínuo que expressa a utilização de um dado recurso como ciclos de CPU. Assumindo que os recursos são limitados, se torna relevante a imposição de uma restrição probabilística (BIRGE; LOUVEAUX, 1997; CHARNES; COOPER, 1963):

$$\mathcal{P}(\mathbf{u} \leq \xi) \geq \alpha \tag{4.11}$$

onde $\mathcal{P}$ é o operador de probabilidade, α é uma probabilidade desejada para que a restrição seja satisfeita e ξ é uma variável de decisão.

Seja $F_{\mathbf{u}}(u)$ a função com a distribuição de probabilidade cumulativa, definida por:

$$F_{\mathbf{u}}(u) = \int_{-\infty}^{u} f_{\mathbf{u}}(z)dz$$

sendo $f_{\mathbf{u}}$ a função densidade de probabilidade. Seja ainda o quantil α definido por:

$$F_{\mathbf{u}}^{-1}(\alpha) = \min\{u : F_{\mathbf{u}}(u) \geq \alpha\}$$

Uma vez que a restrição probabilística (4.11) não é suscetível de otimização direta, se torna necessária a substituição por uma restrição determinística equivalente:

$$\mathcal{P}(\mathbf{u} \leq \xi) \geq \alpha \iff \xi \geq F_{\mathbf{u}}^{-1}(\alpha) \iff F_{\mathbf{u}}^{-1}(\alpha) \leq \xi \tag{4.12}$$

que é uma função da variável de decisão ξ e a constante $F_{\mathbf{u}}^{-1}(\alpha)$, o que configura uma restrição determinística equivalente.

Estamos interessados em modelos onde $\mathbf{u}$ é uma soma controlada de variáveis aleatórias. No modelo de reconfiguração discreto, $\mathbf{x} = (x_{i,k} : (i,k) \in \Omega)$ é um vetor com as variáveis de decisão e $\mathbf{u}(\mathbf{x}) = \sum_{(i,k)\in\Omega} \mathbf{u}_{i,k}x_{i,k}$ é a utilização de todos os servidores, onde $\mathbf{u}_{i,k}$ é uma variável aleatória modelando a utilização do servidor S_i no modo k. No modelo de reconfiguração contínuo, $\mathbf{x} = (x_i : S_i \in \mathcal{S})$ é o vetor de decisão e $\mathbf{u}(\mathbf{x}) = \sum_{S_i\in\mathcal{S}} \mathbf{u}_i x_i$, onde a utilização do servidor S_i é caracterizada por uma variável aleatória $\mathbf{u}_i$ e um nível de utilização x_i. A discussão geral sobre restrições probabilís-

ticas será focada no modelo discreto, porém estes desenvolvimentos são válidos para o modelo contínuo.

A média de $\mathbf{u}(\mathbf{x})$ é $\mu(\mathbf{x}) = \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k}$ com $\mu_{i,k}$ sendo a média de $\mathbf{u}_{i,k}$, qualquer que seja a natureza das variáveis aleatórias elementares. Assumindo-se que as utilizações $\mathbf{u}_{i,k}$ são todas independentes para S_i distintos e, juntamente com a restrição $\sum_{k\in K_i} x_{i,k} = 1$, a variância de $\mathbf{u}(\mathbf{x})$ é $\sigma(\mathbf{x})^2 = \sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2$ com $\sigma_{i,k}^2$ sendo a variância de $\mathbf{u}_{i,k}$.

Na restrição probabilística (4.11), recursos suficientes ξ são instalados para assegurar que a utilização aleatória esteja abaixo da disponibilidade com uma probabilidade α ou maior. Contudo, os recursos são fixos no problema de reconfiguração. Portanto, o escalonador deve decidir sobre os valores das variáveis $\mathbf{x}$ para reconfigurar os servidores de maneira que a probabilidade de utilização induzida $\mathbf{u}(\mathbf{x})$ não exceda a probabilidade α. Isto torna a implementação do equivalente determinístico mais complexa porque o quantil α de $\mathbf{u}(\mathbf{x})$ depende de $\mathbf{x}$.

Porém, se as utilizações dos servidores $\mathbf{u}_{i,k}$ forem caracterizadas por variáveis Gaussianas independentes, então o quantil α se torna independente do vetor de decisão $\mathbf{x}$, uma consequência da soma de variáveis aleatórias Gaussianas ser uma variável aleatória Gaussiana (LEON-GARCIA, 1994, pp. 272). Mais precisamente, $\mathbf{u}(\mathbf{x})$ é caracterizada por uma variável aleatória Gaussiana com média $\mu(\mathbf{x}) = \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k}$ e variância $\sigma(\mathbf{x})^2 = \sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2$. A restrição probabilística assume a forma:

$$\begin{aligned}
\mathcal{P}(\mathbf{u}(\mathbf{x}) \leq \xi) \geq \alpha \iff \mathcal{P}\left(\frac{\mathbf{u}(\mathbf{x}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \leq \frac{\xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \geq \alpha \\
\iff F_{\mathbf{z}}^{-1}(\alpha) \leq \frac{\xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \iff F_{\mathbf{z}}^{-1}(\alpha)\sigma(\mathbf{x}) + \mu(\mathbf{x}) \leq \xi \\
\iff F_{\mathbf{z}}^{-1}(\alpha)\sqrt{\sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2} + \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k} \leq \xi \quad (4.13)
\end{aligned}$$

onde $\mathbf{z} = \frac{\mathbf{u}(\mathbf{x}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$ é uma variável aleatória Gaussiana com média nula e variância unitária, quaisquer que sejam os valores do vetor de decisões $\mathbf{x}$ assumindo que a restrição (4.3c) seja satisfeita. Logo, sob a hipótese que as utilizações $\mathbf{u}_{i,k}$ sejam todas caracterizadas por variáveis aleatórias independentes com distribuição Gaussiana, o quantil α para $\mathbf{u}(\mathbf{x})$ se torna independente de $\mathbf{x}$ e pode ser computado *off-line* e usado no equivalente determinístico. De outra forma, o quantil α teria que ser recalculado para cada $\mathbf{x}$, tornando o equivalente determinístico muito mais complexo do que a desigualdade (4.13).

4.3.2 Modelo Discreto

Considere o problema de reconfiguração discreto em que as utilizações de CPU são variáveis aleatórias. Formalmente, a utilização do servidor S_i no modo k seria caracterizado por uma variável aleatória $\mathbf{u}_{i,k}$. O problema de reconfiguração discreto estocástico pode ser colocado na forma:

$$PD_s: \ fd_s = \max \sum_{(i,k)\in\Omega} A_{i,k}x_{i,k} \tag{4.14a}$$

$$\text{s.a}: \ \mathcal{P}(\sum_{(i,k)\in\Omega} \mathbf{u}_{i,k}x_{i,k} \leq 1) \geq \alpha \tag{4.14b}$$

$$\sum_{k\in K_i} x_{i,k} = 1, \ S_i \in \mathcal{S} \tag{4.14c}$$

$$x_{i,k} \in \{0,1\}, \ (i,k) \in \Omega \tag{4.14d}$$

onde α é uma probabilidade definida pelo usuário/projetista de que a restrição de escalonabilidade seja satisfeita.

A restrição (4.14b) significa que os servidores devem ser reconfigurados nos modos que asseguram que a probabilidade de satisfazer a restrição de escalonabilidade não seja inferior a α.

Hipótese 1 *Todas as utilizações são variáveis aleatórias independentes e cada variável aleatória $\mathbf{u}_{i,k}$ é caracterizada por uma distribuição Gaussiana com média $\mu_{i,k}$ e variância $\sigma^2_{i,k}$.*

Mais adiante, iremos relaxar está hipótese através da modelagem de uma variável aleatória arbitrária com uma variável aleatória Gaussiana que aproxima a distribuição cumulativa da primeira, que será chave para a computação do quantil α e síntese do equivalente determinístico.

Seja $\mathbf{u}(\mathbf{x})$, uma variável aleatória que modela a utilização de CPU, definida por:

$$\mathbf{u}(\mathbf{x}) = \sum_{(i,k)\in\Omega} \mathbf{u}_{i,k}x_{i,k}$$

Sob a Hipótese 1, uma vez que as utilizações dos servidores são independentes, $\mathbf{u}$ tem média $\mu(\mathbf{x})$ e variância $\sigma(\mathbf{x})^2$ definidas por (LEON-GARCIA, 1994):

$$\mu(\mathbf{x}) = \sum_{(i,k)\in\Omega} \mu_{i,k}x_{i,k}$$

$$\sigma(\mathbf{x})^2 = \sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2$$

Assim, a restrição probabilística (4.14b) é também representada como $\mathcal{P}(\mathbf{u}(\mathbf{x}) \leq 1) \geq \alpha$. De acordo com a teoria de programação estocástica (BIRGE; LOUVEAUX, 1997; CHARNES; COOPER, 1963), um equivalente determinístico para esta restrição probabilística é $F_{\mathbf{z}}^{-1}(\alpha)\sigma(\mathbf{x}) + \mu(\mathbf{x}) \leq 1$ se e somente se:

$$F_{\mathbf{z}}^{-1}(\alpha)\sigma(\mathbf{x}) + \mu(\mathbf{x}) \leq 1 \iff F_{\mathbf{z}}^{-1}(\alpha)\sqrt{\sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2} + \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k} \leq 1 \tag{4.15}$$

sendo $\mathbf{z}$ uma variável aleatória Gaussiana com média zero e variância unitária, enquanto $F_{\mathbf{z}}^{-1}(\alpha)$ é o quantil α. Isto leva ao seguinte problema equivalente determinístico voltado à reconfiguração estocástica:

$$\widehat{PD}_s : \ \widehat{fd}_s = \max \quad \sum_{(i,k)\in\Omega} A_{i,k} x_{i,k} \tag{4.16a}$$

$$\text{s.a} : \ F_{\mathbf{z}}^{-1}(\alpha)\sqrt{\sum_{(i,k)\in\Omega} \sigma_{i,k}^2 x_{i,k}^2} + \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k} \leq 1 \tag{4.16b}$$

$$\sum_{k\in K_i} x_{i,k} = 1, \ S_i \in \mathcal{S} \tag{4.16c}$$

$$x_{i,k} \in \{0,1\}, \ (i,k) \in \Omega \tag{4.16d}$$

Note que $fd_s = \widehat{fd}_s$, pois PD_s e $\widehat{PD}_s$ são equivalentes.

Sendo um programa não-linear inteiro, $\widehat{PD}_s$ é geralmente muito mais difícil de ser resolvido que o problema inteiro PD. No sentido de superar este obstáculo, propõe-se o emprego de limites superiores para $\sigma(\mathbf{x})$ o que nos permite empregar os algoritmos desenvolvidos para reconfiguração determinística (ver Seção 4.2.1) na resolução da aproximação do problema de reconfiguração estocástica $\widehat{PD}_s$.

Proposição 1 $\widehat{\sigma}_1(\mathcal{S}) = \sqrt{\sum_{S_i\in\mathcal{S}} \max\{\sigma_{i,k}^2 : k \in K_i\}}$ *é um limite superior trivial para* $\sigma(\mathbf{x})$.

Proposição 2 *Seja* $\widehat{PD}$ *o problema de reconfiguração na forma* (4.3a)-(4.3d) *obtido definindo-se* $u_{i,k} = \mu_{i,k}$ *e* $A_{i,k} = \sigma^2_{i,k}$*, para todo* $(i,k) \in \Omega$*, e trocando o lado direito da restrição* (4.3b) *por* $\left(1 - F_{\mathbf{z}}^{-1}(\alpha)\sigma_{\min}(\mathcal{S})\right)$ *onde*

$$\sigma_{\min}(\mathcal{S}) = \sqrt{\sum_{S_i \in \mathcal{S}} \min\{\sigma^2_{i,k} : k \in K_i\}}$$

Logo, $\widehat{\sigma}_2(\mathcal{S}) = \sqrt{\widehat{fd}}$ *estabelece um limite superior para* $\sigma(\mathbf{x})$*.*

Prova Para qualquer solução factível $\widehat{\mathbf{x}}$ de $\widehat{PD}_s$, é verdade que:

$$\begin{aligned}
\sigma(\widehat{\mathbf{x}}) &\leq \max\{\sigma(\mathbf{x}) : \mathbf{x} \text{ sujeito a (4.16b)}-\text{(4.16d)}\} \\
&\leq \max\{\sigma(\mathbf{x}) : \mathbf{x} \text{ sujeito a (4.16c), (4.16d), e} \\
&\qquad \sum_{(i,k)\in\Omega} \mu_{i,k} x_{i,k} \leq 1 - F_{\mathbf{z}}^{-1}(\alpha)\sigma_{\min}(\mathcal{S})\} \\
&= \sqrt{\widehat{fd}} \\
&= \widehat{\sigma}_2(\mathcal{S})
\end{aligned}$$

dessa forma definindo $\widehat{\sigma}_2(\mathcal{S})$ como um limite superior para $\sigma(\widehat{\mathbf{x}})$. ∎

O limite superior $\widehat{\sigma}_1(\mathcal{S})$ é obtido facilmente, porém o cômputo de $\widehat{\sigma}_2(\mathcal{S})$ envolve a solução de uma versão $\widehat{PD}$ do problema de reconfiguração determinístico. O algoritmo de programação dinâmica e um algoritmo de programação inteira podem resolver tais problemas, porém o custo computacional poder ser alto para instâncias de grande porte. Algoritmos de aproximação tais como o algoritmo de densidade guloso não são opções.

Uma alternativa é a solução da relaxação de programação linear $\widehat{PD}_{\text{LP}}$ de $\widehat{PD}$, que produz um terceiro limite superior $\widehat{\sigma}_3(\mathcal{S}) = \sqrt{\widehat{fd}_{\text{LP}}}$ sendo $\widehat{fd}_{\text{LP}}$ o valor objetivo da relaxação. Uma solução analítica de $\widehat{PD}_{\text{LP}}$ pode ser obtida de maneira semelhante à solução analítica do problema de reconfiguração determinístico PC (ver Seção 4.2.2). Obviamente, $\widehat{\sigma}_2(\mathcal{S}) \leq \widehat{\sigma}_3(\mathcal{S}) \leq \widehat{\sigma}_1(\mathcal{S})$.

Seja $\widehat{PD}_s(\widehat{\sigma})$ o problema $\widehat{PD}_s$ com o termo $\sqrt{\sum_{(i,k)\in\Omega} \sigma^2_{i,k} x^2_{i,k}}$ substituído por um limite superior $\widehat{\sigma}$. O limite superior pode ser um dos limites $\widehat{\sigma}_i(\mathcal{S})$ apresentados acima ou qualquer outro limite superior válido. Além disso, seja $\widehat{fd}_s(\widehat{\sigma})$ o objetivo de $\widehat{PD}_s(\widehat{\sigma})$. Diferentemente do problema de cômputo de limite superior, qualquer algoritmo disponível para resolver PD (OLIVEIRA, 2009) pode ser empregado para resolver o problema de aproximação $\widehat{PD}_s(\widehat{\sigma})$.

Exemplo Ilustrativo

Considere a instância exemplo do problema de reconfiguração discreto determinístico apresentada na Seção 4.2.1. Assuma que $\mu_{i,k} = 0,6u_{i,k}$ e $\sigma^2_{i,k} = 0,1\mu_{i,k}$ para todo $(i,k) \in \Omega$. O limite superior $\widehat{\sigma}_1(\mathcal{S}) = \sqrt{0,1560} = 0,3950$ é obtido imediatamente. Os limites superiores $\widehat{\sigma}_2(\mathcal{S}) = \sqrt{0,084} = 0,2898$ e $\widehat{\sigma}_3(\mathcal{S}) = \sqrt{0,0851} = 0,2917$ são obtidos resolvendo $\widehat{PD}$ como definido na Proposição 2 e através de sua relaxação, respectivamente. Com uma probabilidade $\alpha = 84,13\%$, $F_{\mathbf{z}}^{-1}(\alpha) = 1$ e assim os equivalentes determinísticos para (4.14b) com $\widehat{\sigma}_1(\mathcal{S})$, $\widehat{\sigma}_2(\mathcal{S})$, e $\widehat{\sigma}_3(\mathcal{S})$ são respectivamente:

$$\sum_{(i,k)\in\Omega} \mu_{i,k}x_{i,k} \leq 1 - F_{\mathbf{z}}^{-1}(\alpha)\widehat{\sigma}_1(\mathcal{S}) = 0,6050$$
$$\sum_{(i,k)\in\Omega} \mu_{i,k}x_{i,k} \leq 1 - F_{\mathbf{z}}^{-1}(\alpha)\widehat{\sigma}_2(\mathcal{S}) = 0,7102$$
$$\sum_{(i,k)\in\Omega} \mu_{i,k}x_{i,k} \leq 1 - F_{\mathbf{z}}^{-1}(\alpha)\widehat{\sigma}_3(\mathcal{S}) = 0,7083$$

A Tabela 4.5 apresenta o lado direito do equivalente determinístico para vários níveis de confiabilidade e os três limites superiores $\widehat{\sigma}_i(\mathcal{S})$.

Tabela 4.5: Parâmetros dos equivalentes determinísticos da restrição de escalonabilidade probabilística

		$1 - F_{\mathbf{z}}^{-1}(\alpha)\widehat{\sigma}_i(\mathcal{S})$		
α	$F_{\mathbf{z}}^{-1}(\alpha)$	$\widehat{\sigma}_1(\mathcal{S})$	$\widehat{\sigma}_2(\mathcal{S})$	$\widehat{\sigma}_3(\mathcal{S})$
72,57%	0,6	0,7630	0,8261	0,8250
78,81%	0,8	0,6840	0,7681	0,7666
84,13%	1,0	0,6050	0,7102	0,7083
88,49%	1,2	0,5260	0,6522	0,6499
91,92%	1,4	0,4470	0,5942	0,5916
98,61%	2,2	0,1311	0,3624	0,3582

A Tabela 4.6 mostra os objetivos $\widehat{fd}_s$ e $\widehat{fd}_s(\widehat{\sigma})$ da aproximação $\widehat{PD}_s(\widehat{\sigma})$ para diferentes limites $\widehat{\sigma}_i(\mathcal{S}), i \in \{1,2,3\}$, e probabilidades α. O rótulo "-" indica que uma solução factível não existe para o limite superior e probabilidade dados. O objetivo ótimo $\widehat{fd}_s$ foi computado por meio de enumeração de todas as soluções para $\widehat{PD}_s$.

Tabela 4.6: Objetivos $\widehat{fd}_s$ e $\widehat{fd}_s(\widehat{\sigma})$ para diferentes limites superiores $\widehat{\sigma}_i(\mathcal{S}), i \in \{1,2,3\}$, e probabilidades α

	Objetivo $\widehat{fd}_s(\widehat{\sigma}_i(\mathcal{S}))$			
α	$\widehat{\sigma}_1(\mathcal{S})$	$\widehat{\sigma}_2(\mathcal{S})$	$\widehat{\sigma}_3(\mathcal{S})$	$\widehat{fd}_s$
72,57%	2,248	2,381	2,381	2,381
78,81%	1,996	2,248	2,248	2,248
84,13%	1,863	2,015	2,015	2,113
88,49%	1,632	1,882	1,882	1,996
91,92%	1,401	1,765	1,765	1,882
98,61%	-	1,151	0,909	1,534

4.3.3 Modelo Contínuo

Agora, a utilização de CPU de um servidor S_i é modelada como uma variável aleatória $\mathbf{u}_i$ e um nível de utilização x_i. O desejo de satisfazer a restrição de escalonabilidade com uma probabilidade α é expresso por:

$$\mathcal{P}(\sum_{S_i \in \mathcal{S}} \mathbf{u}_i x_i \leq 1) \geq \alpha \tag{4.17}$$

Hipótese 2 *Todas as utilizações de servidores são variáveis aleatórias independentes, sendo cada variável aleatória* $\mathbf{u}_i$ *caracterizada por uma distribuição Gaussiana com média* μ_i *e variância* σ_i^2.

Como no caso discreto, esta hipótese tem o propósito de simplificar o projeto de algoritmos e será relaxada mais tarde; um procedimento para aproximar uma variável aleatória qualquer com uma variável Gaussiana será desenvolvido na Seção 4.3.4.

Seja $\mathbf{u}(\mathbf{x}) = \sum_{S_i \in \mathcal{S}} \mathbf{u}_i x_i$ uma variável aleatória com a utilização total do processador. A sua média é $\mu(\mathbf{x}) = \sum_{S_i \in \mathcal{S}} \mu_i x_i$. A variância é $\sigma(\mathbf{x})^2 = \sum_{S_i \in \mathcal{S}} \sigma_i^2 x_i^2$, pois as utilizações dos servidores são independentes. Porque as utilizações são Gaussianas sob a Hipótese 2, a utilização total $\mathbf{u}(\mathbf{x})$ é também caracterizada com uma distribuição Gaussiana tendo média $\mu(\mathbf{x})$ e variância $\sigma(\mathbf{x})^2$.

O equivalente determinístico da restrição de probabilidade (4.17) é dado pela seguinte desigualdade:

$$F_{\mathbf{z}}^{-1}(\alpha)\sqrt{\sum_{S_i \in \mathcal{S}} \sigma_i^2 x_i^2} + \sum_{S_i \in \mathcal{S}} \mu_i x_i \leq 1 \iff F_{\mathbf{z}}^{-1}(\alpha)\|(\sigma_i x_i : S_i \in \mathcal{S})\|_2 + \sum_{S_i \in \mathcal{S}} \mu_i x_i \leq 1 \tag{4.18}$$

que consiste de uma restrição cônica de segunda-ordem (*second-order cone constraint*) sob a hipótese que $F_{\mathbf{z}}^{-1}(\alpha) > 0$.

Portanto, o equivalente determinístico do problema de reconfiguração contínuo estocástico consiste de um problema da forma:

$$\widehat{PC}_s : \widehat{fc}_s := \max \; f = \sum_{S_i \in \mathcal{S}} A_i x_i \tag{4.19a}$$

$$\text{s.a} : h := F_{\mathbf{z}}^{-1}(\alpha) \|(\sigma_i x_i : S_i \in \mathcal{S})\|_2 + \sum_{S_i \in \mathcal{S}} \mu_i x_i - 1 \leq 0 \tag{4.19b}$$

$$g_i := x_i - U_i \leq 0, S_i \in \mathcal{S} \tag{4.19c}$$

$$l_i := L_i - x_i \leq 0, S_i \in \mathcal{S} \tag{4.19d}$$

onde f é a função objetivo, h define o equivalente determinístico da restrição de escalonabilidade, e g_i (l_i) são funções que definem os limites superiores (inferiores). Claramente, $\widehat{PC}_s$ é um programa cônico de segunda-ordem (*second-order cone program*, SOCP) (BOYD; VANDENBERGHE, 2004, pp. 156-160), (ALIZADEH; GOLDFARB, 2003). Embora SOCP seja um problema não-linear, algoritmos de tempo polinomial muito eficientes podem ser projetados com base no método de barreira logarítmica (BERTSEKAS, 1995; BOYD; VANDENBERGHE, 2004). Essa é uma consequência da convexidade da função $h(\mathbf{x})$ que define a restrição de escalonabilidade.

O método de barreira aproxima $\widehat{PC}_s$ com um problema irrestrito dado por $\widehat{PC}_s(\epsilon)$, cujo objetivo é:

$$\widehat{fc}_s(\mathbf{x}; \epsilon) := f(\mathbf{x}) + \frac{1}{\epsilon}\phi(\mathbf{x})$$

onde ϕ é a função barreira logarítmica e $\epsilon > 0$ é um parâmetro que controla a qualidade da aproximação.

A função barreira $\phi(\mathbf{x})$ é finita no interior do conjunto factível, porém tende a infinito à medida que $\mathbf{x}$ se aproxima da fronteira de qualquer restrição. Para o problema em consideração, $\phi(\mathbf{x})$ é dada por:

$$\phi(\mathbf{x}) = -\log(-h(\mathbf{x})) - \sum_{S_i \in \mathcal{S}} [\log(-g_i(\mathbf{x})) + \log(-l_i(\mathbf{x}))]$$

A aproximação irrestrita é definida por:

$$\widehat{PC}_s(\epsilon):\ \min_{\mathbf{x}} \widehat{fc}_s(\mathbf{x};\epsilon) := f(\mathbf{x}) + \frac{1}{\epsilon}\phi(\mathbf{x}) \tag{4.20}$$

Seja $\mathbf{x}^\star(\epsilon)$ a solução ótima para o problema $\widehat{PC}_s(\epsilon)$. O conjunto de pontos $\{\mathbf{x}^\star(\epsilon) : \epsilon > 0\}$ é conhecido como caminho central (*central path*) e seus elementos são chamados de pontos centrais (*central points*). Pode-se mostrar que $\mathbf{x}^\star(\epsilon)$ converge a uma solução ótima de $\widehat{PC}_s$ à medida que $\epsilon \to \infty$. Tal estratégia é formalizada no **Algoritmo** 4.3.

Algorithm 4.3 Método de barreira logarítmica para $\widehat{PC}_s$

1: **input:** Strictly feasible $\mathbf{x}$, $\epsilon := \epsilon^{(0)} > 0$, $\rho > 1$, tolerance $\tau > 0$
2: **repeat**$(2n+1)\rho/\epsilon < \tau$
3: 1. *Centering:* obtain $\mathbf{x}^\star(\epsilon)$ by minimizing $f(\mathbf{x}) + \frac{1}{\epsilon}\phi(\mathbf{x})$ starting at $\mathbf{x}$
4: 2. *Update:* $\mathbf{x} := \mathbf{x}^\star(\epsilon)$
5: 3. *Stopping criterion:* quit if $(2n+1)/\epsilon < \tau$
6: 4. *Increase* ϵ: $\epsilon := \epsilon\rho$
7: **until**

Em princípio, qualquer algoritmo de otimização irrestrita pode ser empregado para resolver o passo de centralização. Uma vez que, sob condições não restritivas, a função objetivo $\widehat{fc}_s(\mathbf{x};\epsilon)$ é duas vezes diferenciável e estritamente convexa em $\mathbf{x}$, o método de Newton amortecido se torna uma estratégia adequada pois converge com taxa quadrática à solução ótima na vizinhança de $\mathbf{x}^\star(\epsilon)$. Para detalhes do método de Newton e expressões fechadas do gradiente e matriz Hessiana de ϕ, respectivamente $\nabla\phi$ e $\nabla^2\phi$, sugere-se (BOYD; VANDENBERGHE, 2004, Capítulo 9). A computação de uma solução inicial estritamente factível $\mathbf{x}$, bem como a escolha dos parâmetros $\epsilon^{(0)}$ e ρ são também discutidos nesta referência.

Exemplo Ilustrativo

A instância do problema de reconfiguração com escalonabilidade estocástica foi obtida por meio de uma adaptação da instância determinística descrita na Seção 4.2.2. A variável aleatória $\mathbf{u}_i$ que modela a utilização de CPU do servidor S_i tem uma média $\mu_i = 0,6(U_i + L_i)$ e variância $\sigma_i^2 = 2,5\mu_i$. A solução do problema $\widehat{PC}_s$ para vários níveis de probabilidade α aparece na Tabela 4.7. A solução ótima para cada $\widehat{PC}_s$ foi obtida resolvendo-se um problema de otimização convexo apropriado.

Tabela 4.7: Objetivo ótimo $\widehat{fc}_s$ para diferentes probabilidades α.

α	$F_{\mathbf{z}}^{-1}(\alpha)$	$\widehat{fc}_s$
72,57%	0,6	3,3333
78,81%	0,8	3,2802
84,13%	1,0	3,0464
88,49%	1,2	2,8554
91,92%	1,4	2,6958
98,61%	2,2	2,2160

4.3.4 Aproximando a Utilização de CPU com Variáveis Aleatórias Gaussianas

A teoria apresentada acima pode ser empregada em sistemas onde a utilização de CPU não é caracterizada por variáveis aleatórias Gaussianas. A ideia está na aproximação de uma variável aleatória arbitrária com uma variável aleatória Gaussiana apropriada que superestima a utilização de CPU. Isto significa que a variável Gaussiana induz uma função de probabilidade cumulativa (*cumulative density function*, CDF) que é limitada por cima pela CDF da variável aleatória verdadeira. Em outras palavras, a probabilidade da utilização do processador estar abaixo de um valor ϵ deve ser menor segundo a aproximação Gaussiana do que segundo a variável aleatória verdadeira. Com tais aproximações Gaussianas, os modelos e algoritmos desenvolvidos para escalonabilidade estocástica (discreto e contínuo) podem ser aplicados na reconfiguração de servidores dos sistemas tempo real em consideração. Uma forma de se obter uma aproximação Gaussiana $\mathbf{u}_{i,k}$ consiste em resolver o seguinte problema de otimização não-linear:

$$PG: \min_{\sigma_{i,k},\, \mu_{i,k}} \quad fg = \sum_{i=1}^{n} (F_{\mathbf{u}_{i,k}}(u_j) - F_{\mathbf{z}_{i,k}}(u_j)) \tag{4.21a}$$

$$\text{s.a}: \; F_{\mathbf{u}_{i,k}}(u_j) \geq F_{\mathbf{z}_{i,k}}(u_j),\; j = 1, \ldots, n \tag{4.21b}$$

$$\sigma_{i,k}, \mu_{i,k} \in \mathbb{R}_+ \tag{4.21c}$$

onde:

- $\mathbf{u}_{i,k}$ é a variável aleatória que modela a utilização de CPU;
- $\mathbf{z}_{i,k}$ é a variável aleatória Gaussiana cuja CDF aproxima a CDF de $\mathbf{u}_{i,k}$;
- $F_{\mathbf{u}_{i,k}}(u)$ $(F_{\mathbf{z}_{i,k}}(u))$ é a função densidade de probabilidade cumulativa da variável aleatória $\mathbf{u}_{i,k}$ $(\mathbf{z}_{i,k})$;

- $\{u_j\}_{j=1}^{n}$ é um conjunto de níveis de utilização para os quais a CDF aproximativa deve subestimar a CDF de $\mathbf{u}_{i,k}$; e

- $\mu_{i,k}$ e $\sigma_{i,k}$ são variáveis de decisão, onde a primeira é a média e a segunda é o desvio padrão da aproximação $\mathbf{z}_{i,k}$.

A função objetivo (4.21a) busca minimizar o erro total entre as CDFs, enquanto a restrição (4.21b) assegura que a CDF de aproximação subestima a CDF verdadeira nos pontos amostrais de utilização.

Assume-se que $F_{\mathbf{u}_{i,k}}(u)$ é conhecida ou facilmente estimada a partir de amostras de utilização, o que tornam $\{F_{\mathbf{u}_{i,k}}(u_j)\}_{j=1}^{n}$ constantes dadas para o problema de aproximação PG. Note que a CDF da variável aleatória Gaussiana, especificamente

$$F_{\mathbf{z}_{i,k}}(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi\sigma_{i,k}^2}} \mathrm{e}^{(-\frac{(q-\mu_{i,k})^2}{2\sigma_{i,k}^2})} dq, \tag{4.22}$$

não possui uma solução analítica simples, porém pode ser computada numericamente com um nível de precisão desejado utilizando-se métodos de integração numérica. Com respeito a u, $F_{\mathbf{z}_{i,k}}(u)$ é uma função crescente com uma derivada conhecida na utilização de CPU u. Contudo, as otimizações se darão sobre a média $\mu_{i,k}$ e desvio padrão $\sigma_{i,k}$ que influenciam o formato de $F_{\mathbf{z}_{i,k}}(u)$ e, por sua vez, a qualidade da aproximação.

Encontrar a aproximação Gaussiana ótima exige a resolução de um problema de otimização global, que é uma área avançada especialmente quando envolve funções não-convexas em geral (HORST; PARDALOS; THOAI, 2000; HORST; TUY, 2003). No tentando, a baixa dimensionalidade do espaço de decisão e a evidência numérica indicam que métodos de gradiente descendente podem alcançar soluções satisfatórias. Foi empregado um algoritmo de programação quadrática sequencial (*sequential quadratic programming*, SQP) para resolver PG aproximadamente, no qual os gradientes foram computados numericamente com base na computação numérica da CDF.

A Figura 4.6 ilustra a CDF da utilização de CPU de uma tarefa e a correspondente CDF da variável aleatória Gaussiana aproximativa. As amostras de utilização foram obtidas a partir da evolução de um decodificador de vídeo durante sua aplicação em tempo real em um filme.

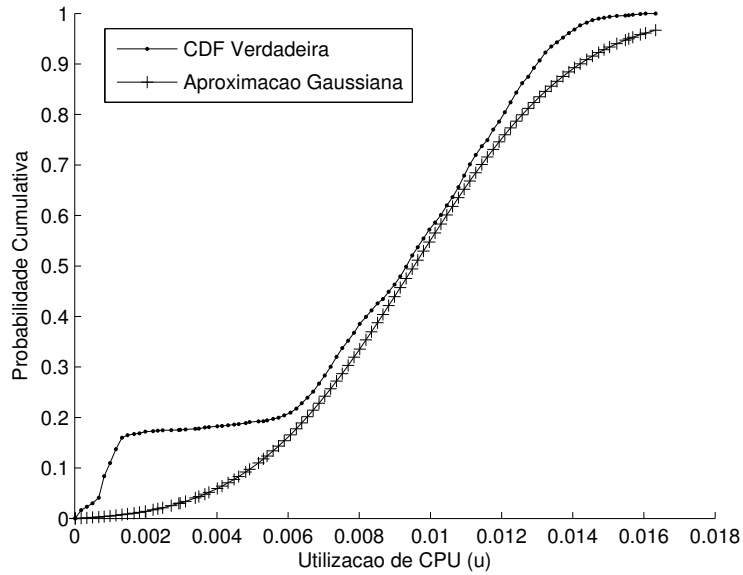


Figura 4.6: Exemplo de CDF da utilização de CPU de uma tarefa e a CDF da variável aleatória Gaussiana de aproximação

4.4 Estudo de Caso

Os mecanismos de reconfiguração dinâmica de servidores são ilustrados por meio de uma aplicação em robótica móvel, considerando desde a definição de estados e modos, a escolha de um modelo de otimização e até a simulação de escalonamento para avaliar os efeitos das escolhas. A aplicação consiste de um robô móvel e seus sistemas de visão, controle e navegação. Mais especificamente, cinco tarefas compõem o sistema robótico: controle de motores; detecção de obstáculos; definição de caminho; manutenção de mapa; e visão. Uma vez que do ponto de vista do escalonador as tarefas de controle de motores e detecção de obstáculos apresentam comportamento idêntico, elas serão agrupadas em uma única tarefa.

4.4.1 Sistemas de Visão, Controle e Navegação de um Robô Móvel

Para ilustrar o modelo de sistema utilizado na aplicação em robótica móvel, considere o software executado em um robô de competição acadêmica. O seu objetivo é percorrer um labirinto sem nenhum conhecimento prévio sobre a disposição de suas paredes. Este software é composto por várias tarefas:

- software de controle em malha fechada para controle dos motores;
- sensores infra-vermelho que detectam as paredes, usando tarefas em *background* para periodicamente atualizar o modelo interno que o robô possui do labirinto;
- sistema de visão que captura objetos de interesse (ícones nas paredes), os catalogando e reconhecendo através do labirinto.

Os quatro sub-sistemas (controle de motores, detecção de paredes, manutenção do mapa/definição de caminho e, finalmente, visão) compartilham o mesmo processador. Portanto, servidores CBS podem ser usados para garantir seu isolamento temporal. Na notação usada neste trabalho, um servidor S_1 poderia receber a tarefa de controle dos motores, S_2 a tarefa de visão e assim por diante.

Cada uma destas tarefas tem múltiplos modos de operação. A tarefa de controle de motores, por exemplo, não necessita de uma reserva de tempo de CPU enquanto o robô está parado. Entretanto, enquanto o robô está em movimento, ela se torna crítica: se um obstáculo é detectado, ela precisa responder rapidamente para garantir que o robô pare em tempo hábil. Pode-se perceber, portanto, que o benefício A_1 para a tarefa atribuída ao servidor S_1 (controle de motores) muda ao longo do tempo de execução do sistema. Enquanto o robô está parado, $A_1 = 0$. Enquanto está em movimento, $A_1 \gg 0$. Estas mudanças em benefício ocorrem em todas as tarefas do robô, porém cada uma delas requer a reconfiguração do escalonador para que a reserva de processador seja redistribuída, maximizando o benefício agregado do sistema.

Outros exemplos de mudança de benefício podem ser encontrados no sistema de visão. Este sistema tem três níveis de análise que devem ser executados para que uma imagem na parede seja reconhecida em sua base de dados. Inicialmente, imagens de baixa resolução são capturadas em alta frequência, calculando periodicamente a chance do quadro conter uma imagem de interesse. Uma vez que esta chance ultrapassa um certo limiar, a tarefa troca de nível para confirmar a presença da imagem. Finalmente, confirmada a presença da imagem, um terceiro nível é ativado. Neste nível final, a imagem é comparada com aquelas armazenadas pelo robô com o objetivo de identificá-la em sua base de dados. A cada mudança de nível, a utilização de processador e a importância (e portanto, benefício) da tarefa crescem. No último destes níveis, a frequência da tarefa é diminuída para um quinto do seu valor original, enquanto a utilização se multiplica por dez.

Seja S_2 o servidor que trata o sistema de visão. Pode-se imaginar que cada um destes níveis leva tanto o benefício A_2 e a utilização requerida U_2 a aumentar. Quando a soma da utilização requerida por todas as tarefas viola a condição (4.2), cabe ao

sistema de reconfiguração decidir qual alocação de tempo de CPU a fazer, sacrificando tarefas menos importantes que trazem menos benefício ao processo.

Como pode-se notar, reconfigurar os parâmetros (Q_i, T_i) dos servidores em $\mathcal{S}$ para maximizar o benefício total do sistema requer a solução de um problema de otimização, no qual a equação (4.2) é uma das restrições. Diversos modelos diferentes deste problema foram criados, cada um com a capacidade de capturar diferentes aspectos de diferentes aplicações como, por exemplo, a natureza discreta ou contínua dos modos de cada uma das tarefas.

4.4.2 Modelo de Escalonamento

Para formalizar a aplicação robótica no contexto da infraestrutura de reconfiguração, é necessário definir os valores de benefício e utilização de cada tarefa para todos os seus possíveis modos de operação. Além disso, as transições que desencadeiam a mudança desses valores precisam ser identificadas, já que serão nesses pontos de transição que uma reconfiguração será requisitada. Os valores apresentados aqui são uma aproximação de valores aferidos de uma aplicação real para o controle de robôs móveis desenvolvida na Universidade Federal de Santa Catarina (ROBOTA, 2009) que usa um sistema simples de execução cíclica no seu escalonamento. Os valores de benefício são definidos com base na importância de cada tarefa para a aplicação como um todo.

Sensor/Atuador

A tarefa sensor/atuador (união das tarefas de controle de motores e detecção de obstáculos) tem apenas um modo de operação que, dependendo do estado do sistema, pode ter dois valores diferentes de benefício e utilização.

Quando o robô está parado, o valor de benefício desta tarefa para o sistema é nulo, uma vez que qualquer reserva de tempo assegurada para ela seria desperdiçado. Quando o robô se move, o seu valor se torna bastante elevado uma vez que a tarefa é crítica para a aplicação; ela é a única capaz de evitar colisões e, se o robô colidir, a competição será perdida. A Tabela 4.8 define os valores para esta tarefa durante ambos estes estados.

Tabela 4.8: Utilizações e valores de benefício para os estados da tarefa sensor/atuador

	Servidor S_1 - Tarefa Sensor/Atuador	
	Parado	Movendo
$k = 1$	$U_{1,1} = 0 \quad A_{1,1} = 0$	$U_{1,1} = 0,1 \quad A_{1,1} = 100$
	$t = \infty$	$t = 0,1s$

Enquanto o robô se move, o valor de $A_1 = 100$ é atribuído ao modo. Este valor será usado como uma marca inicial de criticalidade para as outras tarefas. O fato de ser o único modo para esta tarefa, entretanto, garante que ele será selecionado pelo algoritmo de otimização sempre que estiver disponível.

Definição de Caminho

A tarefa de definição de caminho geralmente se mantém ociosa, já que ela só é ativada quando o robô detecta um obstáculo no seu caminho atual. Enquanto um caminho está sendo percorrido e nenhum obstáculo é detectado, todos os seus modos têm valores de benefício nulo.

Uma vez que um obstáculo é detectado, a definição de um novo caminho a percorrer se torna de grande importância, porém não-crítica. Isto permite a definição de dois modos para esta tarefa, onde o mais valioso tem alta utilização e benefício por ser capaz de calcular um caminho rapidamente. Um modo de menor utilização e benefício também é definido, e quando utilizado leva a tarefa a demorar mais para definir um caminho. A Tabela 4.9 define os valores de utilização e benefício para esta tarefa.

Tabela 4.9: Utilizações e valores de benefício para os estados da tarefa de definição de caminho

	Servidor S_2 - Tarefa de Definição de Caminho	
	Ocioso	Definindo Caminho
$k = 1$	$U_{2,1} = 0 \quad A_{2,1} = 0$	$U_{2,1} = 0 \quad A_{2,1} = 0$
$k = 2$		$U_{2,2} = 0,5 \quad A_{2,2} = 50$
$k = 3$		$U_{2,3} = 1 \quad A_{2,3} = 90$
	$t = \infty$	$t = 0,5s$

Note que o modo $k = 3$ da tarefa de definição de caminho efetivamente ocupará o processador completamente ($U_{2,3} = 1$), mas tem um benefício menor que a tarefa sensor/atuador ($A_{2,3} = 90$, enquanto que para o sensor/atuador, $A_{1,1} = 100$). Isto é uma maneira de atribuir valores relativos entre as tarefas. Uma vez que durante movimentações pelo menos 10% do processador será reservado, o modo $k = 3$ será ativado apenas se o robô se encontrar parado. Também é válido notar que $k = 1$ é um modo de benefício e utilização nulos para esta tarefa. Uma vez que ela não recebe dados externos como as outras, ela pode ser adiada sem perda de dados (como as perdas causadas por sobrecargas no *buffer* de entrada dos sensores ou da câmera, por exemplo). Se esta opção é selecionada pelo reconfigurador, esta tarefa não receberá reserva de banda até pelo menos um novo pedido de reconfiguração.

Manutenção de Mapa

A tarefa de manutenção de mapa se comporta de maneira bastante similar. Existe uma utilização nominal causada pela necessidade de adicionar qualquer novo segmento à representação interna do mapa toda vez que os sensores infravermelho detectam um obstáculo. Porém, periodicamente, a representação interna do mapa se tornará complexa o suficiente para justificar uma simplificação. Isto é feito para liberar memória e acelerar os cálculos de definição de caminhos.

A Tabela 4.10 define os valores de utilização e benefício para a tarefa de manutenção de mapa. Enquanto o mapa se mantém simples, os modos de utilização mais alta não são de interesse. Uma vez que o mapa é complexo o suficiente, simplificá-lo requer mais tempo de processador e traz um benefício maior ao sistema.

Tabela 4.10: Utilizações e valores de benefício para os estados da tarefa de manutenção de mapa

Servidor S_3 - Tarefa de Manutenção de Mapa		
	Nominal	Simplificação Necessária
$k = 1$	$U_{3,1} = 0,1 \quad A_{3,1} = 40$	$U_{3,1} = 0,1 \quad A_{3,1} = 40$
$k = 2$		$U_{3,2} = 0,25 \quad A_{3,2} = 50$
$k = 3$		$U_{3,3} = 0,5 \quad A_{3,3} = 70$
	$t = 1s$	$t = 1s$

Visão

Finalmente, o sistema de visão tem três estados (Nominal, Grosso e Fino), migrando entre eles de acordo com a probabilidade da imagem captada conter um dos símbolos que ele deve reconhecer.

A utilização nominal é baixa, fruto da análise de imagens de baixa resolução capturadas em alta frequência. Os estados seguintes, grosso e fino, necessitam de resoluções cada vez maiores em frequências cada vez mais baixas.

A Tabela 4.11 define esses três estados. Cada um deles tem um modo específico, garantindo a utilização que a tarefa necessita no pior caso e, portanto, levando ao desempenho máximo.

No caso nominal, apenas o processamento em alta frequência de imagens de baixa resolução importa. Quando a análise destas imagens encontra algo de interesse, alcança-se o estado grosso. Se a presença de uma imagem é detectada neste modo, o estado fino é alcançado para que seja realizada a sua identificação.

Tabela 4.11: Utilizações e valores de benefício para os estados da tarefa de visão

Servidor S_4 - Tarefa de Visão			
	Nominal	Grosso	Fino
$k = 1$	$U_{4,1} = 0,25 \quad A_{4,1} = 55$	$U_{4,1} = 0,25 \quad A_{4,1} = 55$	$U_{4,1} = 0,25 \quad A_{4,1} = 55$
$k = 2$		$U_{4,2} = 0,5 \quad A_{4,2} = 75$	$U_{4,2} = 0,5 \quad A_{4,2} = 75$
$k = 3$			$U_{4,3} = 1 \quad A_{4,3} = 95$
	$t = 0,25s$	$t = 0,5s$	$t = 1s$

Uma vez que o processo de identificação de imagens é feito através de um algoritmo de inteligência artificial de alta complexidade, o modo com maior demanda computacional no estado fino é capaz de utilizar 100% do tempo do processador.

O mesmo não pode ser dito do algoritmo utilizado no modo grosso, onde a reserva de uma utilização tão alta levaria ao desperdício de processador. As disponibilidades dos modos $k = 1$ e $k = 2$ no estado fino permitem que a tarefa seja realizada com uma reserva de tempo menor, causando um atraso no seu tempo de resposta.

Discussão

Cada uma destas tarefas pode alcançar qualquer um dos seus estados de forma independente entre si. A utilização da infraestrutura de reconfiguração para definir quanto tempo de processador será alocado a cada tarefa, em tempo de execução e de acordo com seu valor de benefício, libera o projetista do sistema do trabalho de considerar todas as combinações de estados estaticamente.

Enquanto o exemplo desta seção é simples, um sistema complexo com diversas tarefas, estados e modos faria a tarefa de atribuir fatias de processador manualmente extremamente demorada e suscetível a erros. A atribuição de valores para as tarefas, entretanto, é significativamente mais simples e não suscetível à explosão combinatória.

A questão de que modelo matemático usar na reconfiguração deste sistema persiste. Devido à natureza discreta dos modos de cada tarefa, o modelo discreto apresentado na Seção 4.2.1 será utilizado. O uso de um esquema de aproximação permitirá soluções satisfatórias com uma baixa sobrecarga, o que é importante num sistema móvel com capacidade de processamento limitada.

4.4.3 Exemplo de Execução

Para ilustrar o uso da infraestrutura de reconfiguração nesta aplicação, uma linha de tempo com várias migrações entre estados (e, portanto, pedidos de reconfiguração) é apresentada na Figura 4.7 e detalhada a seguir.

Uma simulação do escalonamento desta seção do tempo de execução do robô será apresentada, ilustrando como um número de reconfigurações modifica os parâmetros de escalonamento maximizando o valor agregado do sistema.

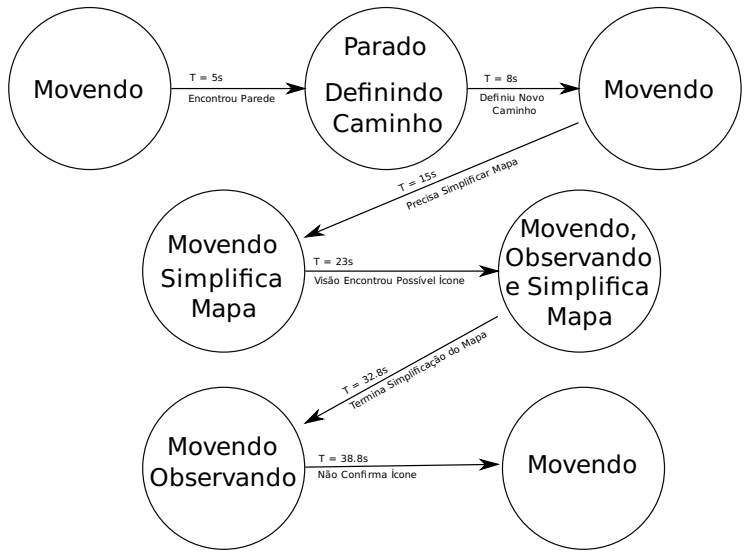


Figura 4.7: Evolução de estados do robô

No início do tempo de vida do robô, ele se encontra parado, com um mapa vazio e sem nenhuma imagem de interesse no seu campo de visão. Sua primeira instrução é para que ele comece a se movimentar; a primeira reconfiguração faz com que os estados e modos para cada tarefa se tornem aqueles mostrados na primeira coluna da Tabela 4.12. Neste estado, a utilização total do sistema é $U = 0,45$.

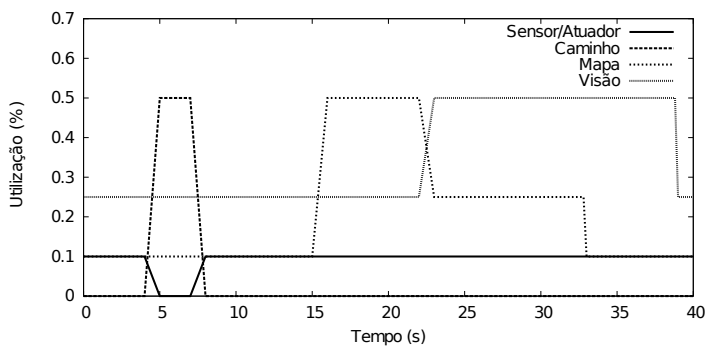


Figura 4.8: Linha de tempo das alocações de orçamento

Tabela 4.12: Mudanças de estado para cada tarefa durante o tempo de execução da aplicação.

	Estado 1 ($t = 0s$)		Estado 2 ($t = 5s$)		Estado 3 ($t = 15s$)		Estado 4 ($t = 23s$)	
Tarefa	Estado	Modo	Estado	Modo	Estado	Modo	Estado	Modo
Sens./Atu.	Movendo	$U_1 = 0,1$	Parado	$U_1 = 0$	Movendo	$U_1 = 0,1$	Movendo	$U_1 = 0,1$
Def. Cam.	Ocioso	$U_2 = 0$	Definindo	$U_2 = 0,5$	Ocioso	$U_2 = 0$	Ocioso	$U_2 = 0$
Mapa	Nominal	$U_3 = 0,1$	Nominal	$U_3 = 0,1$	Simplif.	$U_3 = 0,5$	Simplif.	$U_3 = 0,25$
Visão	Nominal	$U_4 = 0,25$	Nominal	$U_4 = 0,25$	Nominal	$U_4 = 0,25$	Grosso	$U_4 = 0,5$
Total		$U_t = 0,45$		$U_t = 0,85$		$U_t = 0,85$		$U_t = 0,85$

Quando $t = 5s$, depois de algum movimento, o robô detecta uma parede em sua frente. Isto leva a tarefa sensor/atuador a parar, e a tarefa de definição de caminho a alcançar seu estado definindo caminho. Esta mudança de estado leva a uma reconfiguração, que resolve o problema de otimização e redistribui o tempo de processador da forma mostrada na segunda coluna da Tabela 4.12. Nesta mudança de modo, o terceiro modo da tarefa de definição de caminho (com $U_3 = 1$) estava disponível para seleção, porém não foi escolhido porque o estado nominal das tarefas de definição de caminho e de visão tornam essa escolha impossível. Apesar da tarefa de definição de caminho ter a capacidade de ocupar todo o processador, isto nunca ocorrerá devido à alocação mínima das demais tarefas.

Depois de definir o caminho a ser percorrido, o robô começa a se mover novamente em $t = 8s$. Os parâmetros de escalonamento retornam àqueles definidos na primeira coluna da Tabela 4.12. A Figura 4.8 mostra como a infraestrutura de reconfiguração desalocou a utilização previamente reservada à tarefa sensor/atuador em $t = 5s$ e depois a realocou em $t = 8s$, quando o algoritmo de definição de caminho completou seus cálculos.

Em $t = 15s$ o modelo interno do labirinto se torna grande o suficiente para justificar uma simplificação. A tarefa de manutenção de mapa alcança o estado simplificação necessária e pede uma reconfiguração. A alocação de processador se torna aquela mostrada na terceira coluna da Tabela 4.12. Quando $t = 23s$ a tarefa de visão muda seu estado para grosso.

Como pode-se perceber ao contrastar as duas últimas colunas da Tabela 4.12, a infraestrutura de reconfiguração teve de desalocar tempo de processador da tarefa de manutenção de mapa para que o benefício total do sistema fosse maximizado. Isto pode ser atribuído ao benefício mais alto dado a modos com valores de utilização iguais; quando todo o resto é igual e as restrições permitem, o solucionador da otimização sempre escolherá a opção de maior valor de benefício.

Em $t = 32,8s$, a simplificação da representação interna do mapa termina, então U_3 se torna 10%. Em $t = 38,8s$, o sistema de visão para de seguir o que tinha levantado

seu interesse, e U_4 volta aos 25% nominais. A Figura 4.8 mostra estas desalocações de orçamento, e o estado do sistema retorna ao original, da primeira coluna da Tabela 4.12.

A Figura 4.8 mostra como a infraestrutura de reconfiguração faz a administração das reservas de tempo de processador ao longo do tempo de vida do robô, mas um segundo gráfico, mostrando o efeito destas reservas no atraso das tarefas também deve ser analisado. A Figura 4.9 mostra as perdas de *deadline* causadas pela retirada de tempo de processador da tarefa de manutenção de mapa. Até a chegada da tarefa de visão em $t = 23s$, a soma das utilizações no sistema era sempre menor que 100%, portanto não havia perda de *deadlines*. A reconfiguração que realocou tempo de processador da tarefa de manutenção de mapa para a tarefa de visão também teve o efeito de distribuir sobrecarga entre as duas, e, sem ela, a tarefa de visão teria sofrido atrasos mais severos, um maior número de perdas de *deadline* e um tempo de resposta mais longo.

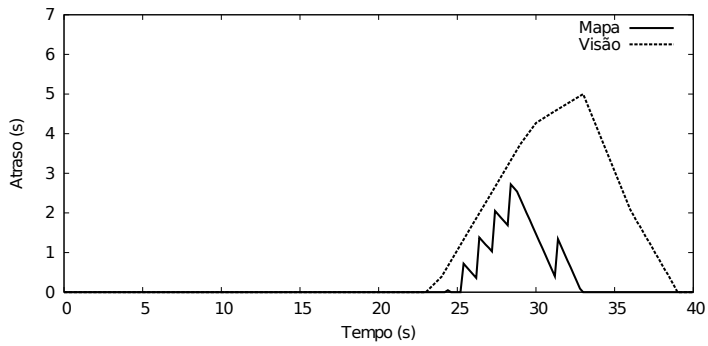


Figura 4.9: Linha de tempo do atraso das tarefas

Este é o propósito da infraestrutura de reconfiguração; quando uma situação de sobrecarga se apresenta, cabe a ela ponderar a importância de cada tarefa para o sistema como um todo e redistribuir tempo de processador entre elas. O efeito disso é mostrado durante o intervalo que se estende do instante $23s$ a $38,8s$ da simulação, onde perdas de *deadline* são compartilhadas entre as tarefas de manutenção de mapa e visão em vez de ocorrer em alguma delas exclusivamente.

4.5 Resumo

Suporte a aplicações de tempo real adaptativas tem sido uma necessidade crescente. Aplicações constituídas de tarefas com múltiplos modos de operação, que devem se ajustar dinamicamente em resposta a eventos não controlados, que estão sujeitas a

grande variações de tempo de execução e a eventuais sobrecargas, imprimem novos requisitos de escalonamento, para os quais deve-se considerar tanto a correção temporal do sistema quanto qualidade de serviço. Este capítulo apresentou algumas soluções que oferecem suporte a reconfiguração dinâmica para tais aplicações. Todas as soluções apresentadas objetivam maximizar o benefício agregado à execução das tarefas da aplicação.

Foram descritos modelos de reconfiguração determinísticos discretos e contínuos, de acordo com os quais os tempos máximo de execução das tarefas são conhecidos. Os modelos discretos são adequados a aplicações constituídas de tarefas que podem executar em um modo de operação dentre um conjunto finito definido pelo projetista. Para os modelos contínuos, o consumo de CPU é ajustado continuamente dentro de um intervalo pré-estabelecido. Algoritmos de aproximação rápidos e eficientes foram projetados para reconfiguração discreta, enquanto um algoritmo exato foi proposto para reconfiguração dinâmica de tarefas com consumo contínuo. O desempenho dos algoritmos desenvolvidos foram avaliados por simulação.

Os modelos discreto e contínuo foram então estendidos para considerar consumo estocástico de CPU, caracterizado por variáveis aleatórias com médias e variâncias conhecidas teoricamente ou experimentalmente. Algoritmos eficientes também foram projetados para reconfiguração dinâmica de tarefas com consumo de CPU estocástico.

Para fins de ilustração, uma aplicação do modelo de reconfiguração determinístico discreto foi considerada para um sistema de robótica móvel.

A construção de suporte adequado a aplicações adaptativas oferece interessantes desafios e possui complexidade. Este capítulo mostrou, usando modelos matemáticos de otimização, possíveis caminhos para se obter soluções efetivas para este problema, contribuindo, desta forma, para este vasto campo de pesquisa.

CAPÍTULO 5

Redes de Sensores: Controle de Acesso ao Meio, Roteamento e Controle de Topologia

Carlos Montez, Flávio Assis e Leandro Buss Becker

Uma *Rede de Sensores Sem Fio* (RSSF) é uma rede desenvolvida para sensoriar ou monitorar um fenômeno específico ou condição de um ambiente e possivelmente atuar sobre ele. Uma RSSF é composta de (tipicamente muitos) *nós sensores* e uma ou mais *estações base*. Um nó sensor é um dispositivo com capacidade de sensoriamento (temperatura, pressão, dentre outros), processamento e comunicação através de canais sem fio. Os dados sensoriados por estes nós são enviados através da rede às estações base, que são responsáveis pelo seu processamento final ou pelo seu encaminhamento a algum sistema de processamento externo à rede. Existem inúmeras áreas de aplicação de RSSF, como, por exemplo, aplicações militares, agricultura de precisão, segurança (em prédios residenciais, centros comerciais, etc.), controle de tráfego e aplicações médicas (controle de medicamentos, controle de acesso de pacientes a locais restritos, etc.) (AKYILDIZ et al., 2002).

Um nó sensor é equipado com quatro unidades básicas: energia, processamento (Unidade Central de Processamento e memória), comunicação sem fio e sensoriamento. Unidades adicionais podem estar presentes, como uma unidade de mobilidade (para permitir o movimento do nó sensor) e uma unidade de localização. Tipicamente

os nós sensores são equipados com processadores de baixo poder computacional, pouca quantidade de memória, relógios de baixa frequência e são alimentados por baterias. Em vários tipos de aplicações, RSSF são utilizadas em locais em que não é possível ou desejável a ação humana direta nos nós após a instalação da rede. Em particular, para estas aplicações a fonte de energia dos nós não é substituível ou recarregável.

O projeto de uma RSSF deve levar em consideração suas características particulares. Algoritmos e arquiteturas desenvolvidos para redes convencionais, como, por exemplo, redes *ad hoc*, em geral não podem ser diretamente aplicados a RSSF por serem baseados em pressupostos não válidos para estas redes (por exemplo, em relação ao padrão esperado de falhas ou às restrições de energia e memória).

Diversos desafios, portanto, surgem relacionados tanto ao projeto e construção quanto à operação de uma RSSF. Exemplos de tais desafios são: o desenvolvimento de protocolos de comunicação que sejam eficientes do ponto de vista energético; protocolos de comunicação que sejam eficientes em relação a interferência, uma vez que altos níveis de interferência podem resultar em maior nível de colisão de mensagens, com consequente aumento no gasto energético e menor vazão e utilização espacial; desenvolvimento de protocolos para garantir o atendimento de níveis de qualidade de serviço na rede, como, por exemplo, garantias de limites máximos de atrasos de transmissão de mensagens fim-a-fim; a criação de metodologias de desenvolvimento de *software* para redes de sensores heterogêneas; e o gerenciamento dos componentes de *software* instalados na rede durante o seu tempo de vida.

Neste capítulo, apresentaremos uma introdução a três importantes problemas relacionados à comunicação em RSSF: controle de acesso ao meio, roteamento e controle de topologia. Em particular, será apresentada uma descrição de abordagens para alguns aspectos específicos destes temas: controle de acesso ao meio para aplicação de RSSF em ambientes industriais; roteamento com restrições temporais; e controle de topologia para difusão de mensagens com eficiência energética.

Os temas controle de acesso ao meio, roteamento e controle de topologia serão tratados, respectivamente, nas seções 5.1, 5.2 e 5.3.

5.1 MAC

Nos dispositivos em RSSF, o envio ou recepção de mensagens usando radiofrequência, mesmo a curtas distâncias, requer significantemente mais energia do que atividades de processamento ou de aquisição de dados em sensores (HAENGGI, 2004). Considerando que a maximização do tempo de vida das redes é um dos principais objetivos

em pesquisas nessas redes, torna-se portanto importante a definição de protocolos e algoritmos de comunicação eficientes.

Em RSSF a comunicação entre dispositivos é usualmente feita através do uso de um canal único e, devido à natureza do meio compartilhado entre os dispositivos, há diversas causas para o consumo desnecessário de energia. A mais conhecida é decorrente de colisões de mensagens, as quais provocam descartes das mensagens envolvidas e eventuais retransmissões que aumentam o consumo de energia. Outros importantes motivos para consumo de energia são: *overhearing*, no qual um dispositivo recebe mensagens destinadas a outros dispositivos; escuta ociosa (*idle listening*), no qual um dispositivo consome energia desnecessariamente escutando um canal vazio; e *overemitting*, no qual um dispositivo envia mensagens para outro que não está pronto para recebê-las. Uma área chave para lidar com todas essas questões é a de escalonamento do acesso ao canal. Essa é uma área que vem sendo, ao longo do tempo, tema de intensa pesquisa e é tratada, essencialmente, pela subcamada MAC (*Medium Access Control*) em redes sem fio.

Técnicas usadas pela subcamada MAC em RSSF envolvem, muitas vezes, o uso de TDMA (*Time Division Multiple Access*) e de um *duty cycle*. O TDMA divide o tempo de acesso ao canal usado pelos dispositivos em compartimentos (*slots*) de tempo, cada qual usado exclusivamente por um dispositivo. Todo dispositivo, antes de enviar mensagens, precisa reservar um compartimento de tempo. Já o *duty cycle* implica na divisão do tempo de comunicação dos dispositivos em períodos de atividade e inatividade. Quanto menor o período de atividade com relação ao de inatividade, mais tempo os dispositivos permanecem inativos e maior a economia de energia, com a contrapartida da redução na taxa de transmissão obtida na rede. Se por um lado o TDMA permite que os dispositivos se organizem de forma a evitar colisões, por outro lado o *duty cycle* complementa essa técnica evitando que um nodo fique ativo simultaneamente com dispositivos inativos, portanto evitando enviar/esperar mensagens para/de estes nodos, evitando assim o desperdício de energia.

Essas técnicas geralmente permitem lidar com os problemas de colisão, *idle listening* e *overemitting*, mas possuem um sobrecusto associado ao envio e processamento de mensagens de controle. Esse sobrecusto pode ser desnecessário em aplicações onde a densidade da rede é pequena ou onde há poucos dispositivos que transmitem simultaneamente. Nesse cenário, protocolos baseados em disputa, como o CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*), são mais adequados. Por este motivo, muitos protocolos de MAC existentes adotam ciclos de envios de mensagens mistos. O padrão IEEE 802.15.4 (COMMITTEE, 2006), por exemplo, define um modo de uso no qual o tempo é dividido em ciclos, delimitados por mensagens de *beacon* en-

viados por um dispositivo coordenador, onde coexistem as três técnicas: disputa pelo canal; acesso garantido ao meio; e período de inatividade. Neste padrão, em cada intervalo entre *beacons* há instantes de tempo onde ocorre disputa pelo canal compartilhado (através de protocolo de acesso ao meio do tipo CSMA-CA), juntamente com instantes de tempo onde o acesso ao meio ocorre sem colisões (através de compartimentos de tempo garantidos) e, também, juntamente com instantes de tempo onde há períodos de inatividade e os nodos podem dormir, tendo seus circuitos do transceiver desligados durante esse tempo, estabelecendo, portanto, um *duty cycle* configurável pelo administrador da rede.

5.1.1 Evolução dos protocolos para redes sem fio

As bases tecnológicas para o desenvolvimento de comunicação sem fio, são antigas. Como exemplo, os primeiros transceivers móveis, conhecidos como *walkie-talkie*, datam do início da década de 1940. No entanto, do ponto de vista dos protocolos MAC para redes sem fio, o ponto de partida foi no início da década de 1970, quando foi feita a primeira transmissão de dados usando o meio sem fio, através do protocolo denominado `ALOHA` (Figura 5.1). As lições adquiridas com as experiências com `ALOHA` deram origem, na década de 1980, a uma série de padrões de redes industriais cabeadas, além da rede de escritório Ethernet (padronizada como IEEE 802.3).

Em 1997 foi criada a tecnologia denominada WiFi (padronizada como IEEE 802.11), a qual forma uma rede composta de equipamentos com interfaces para comunicação de dados sem fio. A criação desta rede abriu possibilidades sem precedentes, permitindo a interligação de equipamentos por meio sem fio, de modo a propiciar a troca de informações entre dispositivos em aplicações residencial, comercial e industrial. Na sequência, no início dos anos 1998, a miniaturização e a queda de preços dos transceivers e de dispositivos sensores tornaram possível o surgimento do paradigma de Redes de Sensores sem Fio.

Muitas das técnicas hoje implementadas em protocolos MAC para RSSF têm como precursor o S-MAC ou Sensor-MAC (YE; HEIDEMANN; ESTRIN, 2002), o qual foi um dos primeiros protocolos de controle de acesso ao meio propostos especificamente para RSSF. Nesse protocolo, dispositivos vizinhos se organizam na forma de agrupamentos (*clusters*) virtuais e se sincronizam para evitar o *overhearing*. Dessa forma, dispositivos desligam seus rádios quando o meio está sendo usado para transmissão de dispositivos de outros agrupamentos. Além disso, o problema de *idle listening* foi tratado forçando que os dispositivos que são vizinhos dentro de um mesmo agrupamento durmam simultaneamente de forma periódica em vez de ficarem escutando um canal

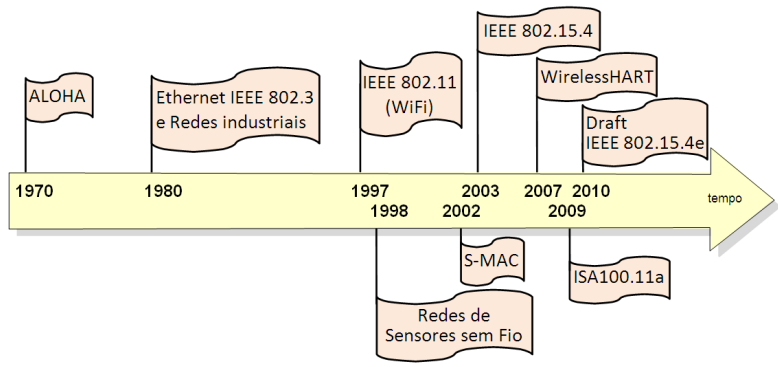


Figura 5.1: Linha de tempo de protocolos e tecnologias de redes cabeadas e sem fio

ocioso.

Também em 2002 foram lançados dois importantes padrões para redes sem fio, o IEEE 802.15.1 que foi baseado nas especificações do Bluetooth e o IEEE 802.16c, padrão para redes sem fio em redes metropolitanas, comercializado sob o nome de WiMAX. Contudo, essas tecnologias não são usualmente empregadas em RSSF.

Em 2003 foi lançada a primeira versão do padrão IEEE 802.15.4, a qual posteriormente sofreu diversas atualizações, sendo a mais importante a efetuada em 2006 e a mais recente efetuada em 2011. Apesar do IEEE 802.15.4 estar sendo adotada pela indústria e também estar se tornando um padrão *de facto* em RSSF, esta especificação de redes da IEEE não foi desenvolvida com esses objetivos específicos. O objetivo primário desta rede LR-WPAN (*Low Power, Low Rate Personal Wireless Networks*) é ser uma infraestrutura para transmissão de dados para distâncias curtas a baixo custo para aplicações que necessitem de uma baixa taxa na comunicação de dados.

Recentemente, a adoção de redes sem fio para ambientes industriais vem sendo intensamente investigada após a consolidação, nesse tipo de ambiente, da tecnologia de Ethernet Industrial (CENA; VALENZANO; VITTURI, 2008). Nos anos de 2007 e 2009 foram lançados, respectivamente, os padrões para redes industriais WirelessHART e ISA 100.11a, os quais tiveram como base para suas camadas de rede inferiores, o IEEE 802.15.4. Em 2010 foi lançado um *draft* do IEEE 802.15.4 voltado especificamente para o ambiente industrial, denominado IEEE 802.15.4e. Devido à importância da adoção de RSSF em aplicações industriais, principalmente no que concerne o desenvolvimento de técnicas e abordagens para a subcamada MAC, iremos tratar na sequência sobre esse tema.

5.1.2 Protocolos MAC e redes sem fio para automação industrial

Redes industriais são aquelas especificamente projetadas para operação com restrições temporais e com requisitos de tolerância a faltas (*fault tolerance*), para serem usadas predominantemente em aplicações de controle e automação. Devido às restrições de tempo real, até recentemente essas redes eram usualmente cabeadas e implementadas somente as camadas um (Física), dois (Enlace) e sete (Aplicação) do modelo de referência OSI, evitando-se, assim, a comunicação com múltiplos saltos e atrasos associados ao enfileiramento nos nodos intermediários (WILLIG, 2008). Contudo, as redes industriais cabeadas apresentam pouca flexibilidade na mudança de suas configurações e um alto custo de implantação e manutenção. Essas ligações podem custar centenas de dólares por metro nas aplicações convencionais, chegando a custar até milhares de dólares por metro, no caso da necessidade de fiação especializada para ambientes agressivos (INDUSTRIAL WIRELESS COMMUNITY, 2002).

Os recentes avanços nas redes sem fio e desenvolvimento de componentes de baixo custo e fácil implantação, tornaram possível substituir-se gradualmente as redes cabeadas industriais pela alternativa sem fio, permitindo se eliminar dezenas de milhares de metros de fiação na indústria. Além dessa vantagem, em ambientes industriais, pequenos nodos sensores podem ser acoplados em pontos estratégicos das máquinas objetivando monitorar o estado das mesmas, no sentido de verificar se estão sujeitos a fadigas ou operando fora das suas especificações, aumentando sua disponibilidade e vida útil.

A introdução da tecnologia de redes sem fio em ambientes industriais pode ser feita gradualmente tornando necessária sua coexistência com as tecnologias de redes cabeadas. No entanto, mesmo para essa implantação gradual, atualmente há pelo menos três grandes problemas a serem resolvidos (CENA; VALENZANO; VITTURI, 2008): (i) como o meio é compartilhado por todos os dispositivos, a vazão total obtida por cada dispositivo geralmente é muito pequena. Exacerbando este problema, ainda existem os sobrecustos introduzidos pelos protocolos MAC das redes sem fio; (ii) técnicas de acesso ao meio baseadas em disputa (ex. CSMA/CA), usadas frequentemente em redes sem fio, introduzem atrasos máximos que não podem ser pré-determinados; e (iii) há sérios problemas na confiabilidade dessas redes pois os canais de redes sem fio são muito mais sujeitos a erros do que os cabeados.

Uma estratégia para lidar com esses três problemas mencionados é aumentar a diversidade espacial/frequencial/temporal dos protocolos MAC de redes sem fio que executam nesses ambientes. A diversidade em redes sem fio – algumas vezes referida como diversidade de canal ou diversidade de enlace – está relacionada com o fenô-

meno em que transmissões por diferentes canais, faixa de frequência ou instantes de tempo, possuem diferentes condições de recepção, e sofrem diferentes atenuações e perdas. Adotar uma estratégia de diversidade reduz os problemas de vazão, previsibilidade temporal e de confiabilidade relatados em (CENA; VALENZANO; VITTURI, 2008), atendendo melhor os requisitos de tolerância a faltas desses ambientes industriais (CHEN; ZHANG; MARSIC, 2009).

Diversidade Temporal. Como a comunicação sem fio é sujeita a falhas de transmissão – devido a colisões de mensagens, interferências ou atenuação no sinal –, a retransmissão de mensagens (ARQ - *Automatic Repeat reQuest*) é uma técnica fundamental utilizada pela maioria dos protocolos MAC. Algoritmos que usam um tempo de *backoff* antes de retransmitir a mensagem tentam exatamente obter uma maior diversidade temporal, espalhando no tempo a retransmissão da mensagem, buscando aumentar a chance de sucesso. No entanto, há limites superiores para os tempos de *backoff*, pois em aplicações industriais geralmente há necessidade de se estabelecer soluções de compromisso entre atender as restrições temporais das aplicações (ainda que sejam deadlines *soft*) e aumentar a taxa de sucesso com a consequente economia de energia, espalhando o máximo possível os tempos de *backoff* das mensagens, evitando, assim, possíveis retransmissões futuras. Outra abordagem possível – comum na área de telecomunicações mas que é raramente usada em protocolos MAC de redes industriais – é empregar uma codificação de correção de erros em avanço (FEC - *Forward Error Correction*), espalhando a informação sobre um período mais longo de tempo de transmissão.

Diversidade em Frequência. No caso de erros por interferência no sinal, a diversidade temporal deve ser empregada em conjunto com a diversidade em frequência. A propagação de sinais em diferentes frequências experimenta diferenças com relação à reflexão, difração e espalhamento, ainda que seja considerado o mesmo instante de tempo e localização. Medidas corretivas para esse tipo de desvanecimento incluem transmissões simultâneas por múltiplas subportadoras e salto entre frequências (FHSS - *Frequency Hopping Spread Spectrum*). Como vantagem adicional ao aumento na confiabilidade, essas técnicas podem melhorar a vazão da rede, permitindo que dispositivos adjacentes transmitam simultaneamente em diferentes faixas de frequência.

Diversidade Espacial. Entre um transmissor e um receptor podem existir múltiplos percursos para o sinal se propagar. Para aumentar a diversidade espacial, no nível

físico, múltiplas antenas podem ser usadas, e na camada de rede, uma topologia em malha (*mesh*) na organização da rede também aumenta a sua confiabilidade.

Adotar uma abordagem para aumentar a diversidade – seja temporal, frequencial ou especial – como um esquema para melhorar o transporte de dados, pode funcionar em qualquer camada ou numa combinação das mesmas. Contudo, importante destacar que esquemas que envolvem a diversidade na camada física, influenciam também os protocolos da subcamada MAC, devido à proximidade entre essas duas camadas.

5.1.3 IEEE 802.15.4

A especificação do IEEE 802.15.4 (COMMITTEE, 2006) define as camadas físicas e subcamada MAC da pilha de protocolos para redes LR-WPAN. O padrão especifica dois tipos diferentes de dispositivos para essas redes: FFD (*full-function device*) e RFD (*reduced-function device*). A diferença entre eles reside no fato que, enquanto um dispositivo RFD opera com uma implementação mínima do protocolo, e pode atuar apenas como um dispositivo simples, um nodo FFD pode atuar também como um coordenador da rede (*PAN coordinator*), provendo serviços de sincronização para rede, geralmente através do envio de mensagens de *beacon*. Por esse motivo, uma rede IEEE 802.15.4 necessita ter pelo menos um dispositivo atuando como nodo coordenador, gerenciando os outros nodos da rede. Contudo, mesmo para o coordenador da rede, o envio de *beacons* não é obrigatório porque o MAC do IEEE 802.15.4 suporta dois modos de operação: com e sem *beacon* (Figura 5.2).

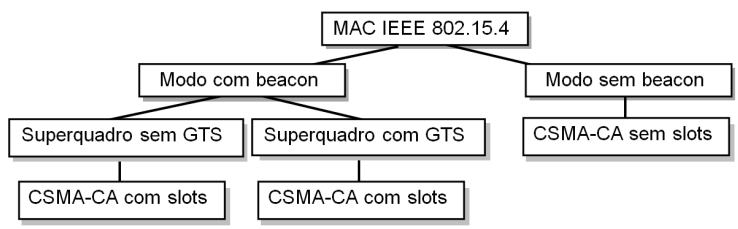


Figura 5.2: Modos de operação do MAC em redes IEEE 802.15.4

O protocolo padrão usado para o controle de acesso ao meio é o CSMA/CA, e duas versões são oferecidas pelo padrão: com e sem slots. O modo CSMA-CA com slots pode ser usado apenas no modo com beacon, e a principal diferença entre os dois modos reside no fato que no modo CSMA-CA com slots, a contagem de slots de *backoff* de um dispositivo desta rede necessita estar alinhado com o início de um *beacon* de transmissão.

A Figura 5.3 ilustra um exemplo de uma rede IEEE 802.15.4 configurada no modo CSMA-CA com *beacon*. Nela é possível se observar a importância de dois parâmetros: *Beacon Order* (BO) e *Superframe Order* (SO). O parâmetro BO define o intervalo de transmissão do *beacon*, ou seja, o *Beacon Interval* (BI); enquanto SO determina a duração de tempo em que o superquadro fica no modo ativo, ou seja, o *Superframe Duration* (SD). No exemplo mostrado, o valor de BO é igual a $SO + 1$. Esses valores causam a existência de período de tempo inativo, no qual dispositivos não podem transmitir. Em outras possíveis configurações de rede, os valores de BO e SO podem ser iguais, não existindo período inativo, com o intervalo entre *beacons* coincidindo com a duração do superquadro.

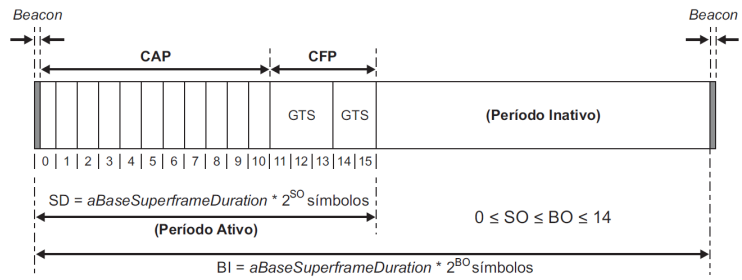


Figura 5.3: Um superquadro do IEEE 802.15.4 com GTS e período inativo

Adicionalmente ao *Contention Access Period* (CAP), onde os dispositivos transmitem disputando o acesso ao meio usando o protocolo CSMA/CA, um superquadro pode, opcionalmente, possuir um período sem disputa pelo meio – o *Contention Free Period* (CFP). O CFP é usado por dispositivos que necessitam garantias de que não haverá colisões de suas mensagens com outros dispositivos IEEE 802.15.4. Este período de tempo consiste na alocação, pelo dispositivo coordenador da rede, de compartimentos garantidos – *Guaranteed Time Slots* (GTS). Supostamente, a alocação de compartimentos garantidos, através da definição de um período de tempo sem disputas, resolveria o problema das aplicações que necessitam de garantias tempo real. Entretanto, uma grande desvantagem deste mecanismo é que apenas sete compartimentos podem ser alocados para todos os dispositivos, o que é insuficiente para a maioria das aplicações.

5.1.4 WirelessHART

A tecnologia WirelessHART (*Wireless Highway Addressable Remote Transducer*) (KIM et al., 2008) permite um protocolo de comunicação bidirecional que prevê a troca de

dados entre instrumentos inteligentes de campo e sistemas de coleta numa planta industrial. Seus transceivers funcionam na banda 2.4 GHz mantendo compatibilidade na camada física com redes IEEE 802.15.4. No entanto, sua subcamada MAC usa a técnica TDMA e salto entre frequências para controle de acesso ao meio. Esta técnica de MAC permite a comunicação determinista e livre de colisões entre dois dispositivos, cada um associado a um compartimento: um como fonte e outro como destino. Os compartimentos migram de uma frequência a outra, segundo uma ordem determinada, aumentando a diversidade em frequência.

Os dispositivos de comunicação são associados a um superquadro, um compartimento dentro de um superquadro e um canal. Este trio forma um enlace de comunicação entre dois dispositivos. A camada de enlace de dados comporta a transferência de dados de uma origem a um destino, com retransmissão automática para garantir a ausência de erros na transferência de dados. Nesta camada estão implementados serviços de QoS, divididos em duas classes: com prioridade e com limite de tempo. As prioridades são classificadas em quatro níveis: comandos, dados de processos, normal e alarme. Os limites de tempo são dados na própria mensagem que especifica em cada solicitação limites superiores sobre a duração do tempo máximo permitido para a conclusão de cada instância da tarefa.

5.1.5 ISA 100.11a

Uma rede ISA100.11a (ISA, 2008) consiste de uma coleção de dispositivos físicos que se comunicam através de enlaces compatíveis com o padrão IEEE 802.15.4. Enquanto no WirelessHART todos os dispositivos de campo são roteadores com capacidade de repassarem pacotes para outros dispositivos, no ISA100.11a os dispositivos sensores e atuadores não tem capacidade de roteamento. Dessa forma, neste padrão os instrumentos de campo precisam ser definidos *a priori* ou como nodos finais, sem capacidade de rotear pacotes, ou como nodos roteadores (PETERSEN; S., 2011). Essa flexibilidade permite reconfigurar redes ISA100.11a suportando topologias em malha, agrupamento em árvore ou estrela

A exemplo do WirelessHART, o ISA100.11a utiliza na camada física os canais 11 a 26 do padrão IEEE 802.15.4. Da subcamada MAC do IEEE 802.15.4, utiliza algumas funcionalidades. No entanto, em vez de adotar os mecanismos padrão de retransmissão e de tempo de `backoff`, o ISA100.11a implementa seu próprio mecanismo de retransmissão, envolvendo diversidade espacial e em frequência.

Como esquema para aumentar diversidade frequencial, saltos entre canais são cuidadosamente programados para, a cada salto, ocuparem um novo canal do IEEE 802.11.

Além disso, como a sequência de saltos é pré-determinada pelo administrador da rede, no caso de haver várias sub-redes, cada uma pode utilizar uma sequência defasada de seu vizinho para evitar interferências mútuas.

O ISA100.11a suporta tanto compartimentos de tempo dedicados para tráfego regular e previsível como compartimentos de tempo compartilhados para alarmes, tráfego em rajadas e retransmissão. Para cada compartimento é definido quem é o transmissor e quem é o receptor e, portanto, um dispositivo saberá em que momento deve ouvir o meio para receber um mensagem e em que momento deve transmitir para um outro determinado dispositivo.

5.1.6 IEEE 802.15.4e

Apesar do padrão IEEE 802.15.4 estar sendo aceito cada vez mais como um padrão industrial, este não foi originalmente desenvolvido para tratar aspectos de tempo real (CHEN; GERMAN; DRESSLER, 2010). Por este motivo, devido à influência dos mecanismos já introduzidos no WirelessHART e no ISA100.11a, há propostas de estender este padrão incorporando, dentre outras, as seguintes características:

- *Comunicação com baixa latência*: para reduzir os atrasos fim a fim na entrega de mensagens, visando principalmente aplicações de controle;
- *Estrutura com múltiplos superquadros e escalonamento TDMA*: permitindo a alocação de dezenas de compartimentos, contornando as limitações impostas pelo padrão original de se poder alocar apenas sete GTS a cada superquadro; e
- *Salto entre canais*: para aumentar a diversidade de frequência, adicionando robustez e permitindo a coexistência com outras redes sem fio.

Nesse sentido, essas propostas estão sendo consolidadas atualmente no padrão, IEEE 802.15.4e (COMMITTEE, 2010), o qual foi proposto como uma emenda ao padrão IEEE 802.15.4, e atualmente aprovado na forma de um *draft*, em 2010, destinado a atender à demanda industrial.

Como exemplo de mecanismo para suportar comunicação com baixa latência entre o coordenador PAN e seus dispositivos sensores/atuadores, em redes com topologia estrela, propõe-se um mecanismo opcional de quadro MAC curto, com apenas 1 octeto de cabeçalho, permitindo a redução tanto no tempo de processamento do quadro quanto no tempo de sua transmissão. O acesso físico passa a ser regido por um esquema TDMA com compartimentos (*time slots*) definidos em um superquadro de tamanho fixo. Esses

compartimentos podem ser dedicados para dispositivos sensores, atuadores e também para o gerenciamento da rede, sendo acessados diretamente sem uso de CSMA-CA (portanto, sem a necessidade de escutar o meio antes de transmitir).

Uma estrutura com múltiplos superquadros pode ser definida de forma opcional. Esta estrutura, denominada *multi-superframe*, é formada por um ciclo de superquadros, cada qual composto por um quadro de *beacon*, período com contenção (CAP) e período sem contenção (CFP). Além dos parâmetros BO e SO, já existentes no padrão IEEE 802.15.4 e que servem para definir o intervalo entre beacons e a duração do superquadro, respectivamente, um novo parâmetro *MacMulti-superframeOrder* (MO) é usado para definir o tamanho do *multi-superframe*.

Durante um período com contenção os nodos precisam ficar ativos, pois podem necessitar receber quadros de outros nodos. Portanto, com objetivo de permitir economia de energia, um mecanismo denominado Redução de CAP (*CAP Reduction*) pode ser usado em múltiplos superquadros. Caso esse mecanismo seja usado, excetuando-se o primeiro superquadro, todos os superquadros subsequentes no *multi-superframe* terão apenas períodos sem contenção (CFP).

O mecanismo de reconhecimento múltiplo, conhecido como *Group ACK* pode ser usado para melhor gerenciar as mensagens de reconhecimento em aplicações com controle centralizado (ex. aplicações de controle via rede), evitando-se o envio de mensagens de reconhecimento individuais para cada dispositivo. Um *Group ACK* é um compartimento reservado pelo coordenador PAN, contendo um bit correspondente a cada dispositivo, e que serve para informar a estes, se seus quadros de dados foram corretamente recebidos ou não. Propõe-se, junto com este mecanismo, a alocação de compartimentos dedicados para que os dispositivos que não tiveram sucesso em suas tentativas anteriores façam suas retransmissões.

Para aumentar a diversidade em frequência, o modo de salto de canais pode ser usado dentro de um superquadro em uma estrutura *multi-superframe*. Como exemplo, cada compartimento dentro de um superquadro pode usar um canal distinto, escolhendo um dentre os 11 canais disponíveis, evitando assim interferência de outros dispositivos que estejam no alcance do rádio. Dentro de um *multi-superframe*, a sequência de saltos entre canais precisa ser pré-definida, mas a escolha cuidadosa da sequência de saltos permite introdução de um deslocamento (*offset*) em frequência em compartimentos de superquadros contíguos, introduzindo uma ortogonalidade entre o tempo e a frequência, o que ajuda a reduzir a possibilidade de interferências.

5.2 Protocolos de Roteamento para RSSF

O roteamento de mensagens em RSSF se faz necessário quando o nodo destinatário não está ao alcance do nodo transmissor, fazendo com que a mensagem seja encaminhada através de nodos intermediários denominados *hops*. Esta tarefa é de responsabilidade da *Camada de Rede*. A fim de entendermos as peculiaridades dos protocolos de roteamento para RSSF, é necessário lembrar que seus nodos são caracterizados por restrições em termos de capacidade de processamento, largura de banda e disponibilidade de energia. De um modo geral, as técnicas de roteamento utilizadas em RSSF buscam maximizar a eficiência energética do sistema a fim de tentar prolongar ao máximo o tempo de vida da rede, sempre levando em consideração as restrições mencionadas.

Conforme destacado em (AL-KARAKI; KAMAL, 2004), os protocolos de roteamento podem ser classificados de acordo com a **estrutura da rede** , com o **modo de operação** dos protocolos ou ainda em **categorias**. São três as estruturas de rede típicas: plana, hierárquica ou baseada em posicionamento. Nas redes planas todos os nodos têm o mesmo papel, enquanto em redes hierárquicas os nodos são agrupados em aglomerados denominados *clusters*, onde alguns nodos especiais exercem o papel de coordenação. Já nas estruturas baseadas em posicionamento, são utilizadas informações de localização para direcionar o encaminhamento das mensagens ao invés de mandar para toda a rede. Um protocolo é considerado *adaptativo* se puder controlar certos parâmetros para se adaptar a diferentes condições de funcionamento da rede e também de quantidade de energia disponível.

Considerando a classificação por **modo de operação**, existem os algoritmos de roteamento que consideram *múltiplos caminhos* (multipath routing), os quais buscam uma maior eficiência em contraponto aos algoritmos tradicionais, que têm um único caminho. Além disso, também melhora o aspecto de tolerância a falhas (resiliência) desses protocolos, dado o fato de que são previstas rotas redundantes entre determinadas origens e destinos. Evidentemente há um aumento no número de mensagens trocadas, gerando uma maior sobrecarga na rede. Por sua vez, os algoritmos de roteamento *baseados em consulta* (query-based routing) assumem que o nodo de destino envia uma consulta de dados pela rede, e o(s) nodo(s) que possuir o dado envia uma resposta. Relacionados são os protocolos *baseados em negociação*, os quais usam descritores de dados de alto nível para caracterizar os dados trocados, o que ajuda a diminuir a redundância na troca de dados. Além disso, há uma constante negociação entre os nodos para evitar o envio de mensagens desnecessárias ou redundantes. Já os protocolos baseados em QoS, o critério de economia de energia deve ser balanceado com a quali-

dade da rede. Os requisitos típicos de qualidade são atraso de entrega (delay), limites de entrega (deadline), largura de banda, entre outros. Finalmente, existe a classificação baseada em processamento de dados, podendo ser coerente ou não-coerente. No modelo não-coerente, o nodo realiza localmente o processamento do dado recebido. Já no roteamento coerente, os dados são enviados com um processamento mínimo ou até mesmo sem processamento.

Por fim, os protocolos de roteamento também podem ser classificados em *categorias*, dependendo do modo como o nodo de origem encontra o destino da informação. São três as categorias de protocolos: proativos, reativos e híbridos. Nos protocolos *proativos* (também chamados de baseados em tabelas) as rotas são computadas antecipadamente, enquanto que nos protocolos *reativos* as rotas são calculadas sob demanda. Já os protocolos *híbridos* se baseiam na combinação de ambas as técnicas.

5.2.1 Visão Geral

Os protocolos desenvolvidos para redes planas geralmente partem do pressuposto que existe um grande número de nodos na rede, sendo portanto inviável estabelecer um identificador global para cada nodo. Além disso, esses protocolos vislumbram um roteamento do tipo *data-centric*, onde a BS envia consultas para regiões específicas da rede e aguarda os dados dos sensores presentes em tais regiões. Uma vez que os dados são enviados como resposta a consultas, é necessário utilizar uma nomenclatura baseada em atributos para especificar as propriedades desejadas dos dados. Os primeiros trabalhos nesta linha são o *SPIN* (HEINZELMAN; BALAKRISHNAN, 1999; KULIK; HEINZELMAN, 2002) e o *Direct Diffusion* (INTANAGONWIWAT; GOVINDAN; ESTRIN, 2000), os quais comprovaram ser possível economizar energia através da negociação entre nodos e da eliminação de dados redundantes. Estes trabalhos motivaram o surgimento de diversos outros trabalhos nesta linha, conforme detalhado em (AL-KARAKI; KAMAL, 2004).

O roteamento hierárquico, também chamado "baseado em cluster", possui vantagens em relação a escalabilidade da rede e a eficiência de comunicação. Além disso, eles também são energeticamente eficientes. A idéia por trás desses protocolos é que existem dois tipos de nodos: nodos comuns e nodos com maior capacidade. Enquanto os nodos comuns são responsáveis apenas por realizar monitoramento, os nodos maiores realizam a coleta e o encaminhamento dos dados. Essa organização ajuda a reduzir o número de mensagens trocadas, além de contribuir para a escalabilidade, o tempo de vida e a eficiência energética do sistema. A organização é feita em duas etapas (ou camadas), sendo a primeira responsável por escolher os nodos coordenadores, sendo

elegíveis os nodos de maior prioridade, e a segunda por realizar o roteamento propriamente dito. Observa-se, entretanto, que a maioria das abordagens nessa direção não trata de roteamento propriamente, mas sim da escolha dos coordenadores, o que pode ser considerado uma questão ortogonal ao roteamento. Os principais protocolos hierárquicos são o LEACH (HEINZELMAN; CH; BALAKRISHNAN, 2000), o PEGASIS (LINDSEY; RAGHAVENDRA, 2002) e o TEEN/APTEEN (MANJESHWAR; AGRAWAL, 2001, 2002).

Por sua vez, os algoritmos de roteamento baseados em posicionamento (ou geográficos) realizam o encaminhamento de mensagens com base na localização dos nodos. A obtenção das coordenadas dos nodos pode ser obtida através de estimativas, por exemplo utilizando a força do sinal RF, ou através do uso de GPS de baixo consumo (que acaba encarecendo os nodos). A fim de economizar energia, tais técnicas costumam colocar os nodos em modos de dormência caso não haja atividades. Quanto mais nodos em estado de dormência, maior a economia de energia. Os principais algoritmos de roteamento geográficos são o GAF (XU; HEIDEMANN; ESTRIN, 2001), o GEAR (YU; GOVINDAN; ESTRIN, 2001) e MFR/GDIR (STOJMENOVIC; LIN, 1999).

No projeto AST são tratadas questões relacionadas com os sistemas de automação, os quais de maneira em geral possuem requisitos de tempo real. De fato, esse tem sido o ponto de investigação constante na parte de roteamento de mensagens. Desta forma, na próxima seção são apresentados os trabalhos relacionados ao provimento de requisitos temporais de qualidade de serviço (QoS). Posteriormente, apresenta-se a proposta criada para prover roteamento em problemas dessa natureza.

5.2.2 Roteamento Geográfico com Provimento de QoS

Dentre os vários protocolos de roteamento que lidam com QoS (Qualidade de Serviço), os protocolos de roteamento geográfico são os que apresentam o melhor desempenho no aspecto de entrega de mensagens e cumprimento de requisitos temporais, além de prover escalabilidade (JIN et al., 2009). Estes protocolos são também caracterizados pelo seu aspecto guloso[1], sempre escolhendo o menor caminho entre a fonte e o destino. Essa característica não é exatamente uma virtude, pois tende a gerar congestionamentos nos caminhos escolhidos, além de poder causar esgotamento de bateria dos nodos que fazem parte do menor caminho, visto que eles estarão permanentemente ocupados repetindo mensagens.

As próximas subseções descrevem os principais trabalhos envolvendo protocolos de roteamento geográficos para RSSF com necessidade de requisitos temporais de QoS.

[1] do termo inglês "*greedy*"

Protocolo RAP

Em (LU et al., 2002) os autores propuseram a chamada Arquitetura de Comunicação para RSSF de Larga Escala (RAP). Este foi o primeiro estudo detalhado sobre o cumprimento de prazo em uma rede de sensores com múltiplos saltos.

A arquitetura do RAP é composta por cinco componentes: (i) o componente *Query/Event Service APIs* é uma interface entre a aplicação de sensoriamento e as camadas inferiores do protocolo; (ii) o componente *Location Addressed Protocol* (LAP) é responsável pelo provimento de três tipos de comunicação: *unicast*, *area multicast* e *area anycast*; (iii) o componente *Geographic Forwarding* (GF) *routing protocol* é responsável por fazer o roteamento guloso dos pacotes; (iv) o componente *Velocity Monotonic* (*packet*) *Scheduling* (VMS) é responsável por escalonar os pacotes de acordo com o prazo e a distância do nodo fonte ao *sink* e (v) o componente MAC, o qual é priorizado para comportar a política definida pelo VMS.

A arquitetura de comunicação RAP provê um bom desempenho no cumprimento de prazos, entretanto os experimentos realizados não levaram em conta uma grande carga na rede, i.e. com vários nodos podendo transmitir mensagens. Além disso o RAP não foi projetado para ambiente de mobilidade. Estas características se encontram presentes no cenário da corrida. Ademais, a única política de escolha de rota é feita puramente pela posição geográfica dos nodos, o que pode gerar congestionamento em caso de larga escala de nodos fontes.

Protocolos SPEED e MMSPEED

O protocolo SPEED (HE et al., 2003) é um dos mais importantes protocolos de roteamento com garantias de qualidade de serviço para aplicações *soft real-time*. O SPEED cunhou a estratégia de velocidade[2] de transmissão dos pacotes, o que influenciou vários protocolos desde então. O protocolo requer a manutenção de uma tabela de vizinhança pela troca periódica de mensagens. O atraso na troca de mensagens com o recebimento do ACK é computado para formar a noção de velocidade de transmissão. Assim, os vizinhos que têm o menor atraso são escolhidos para retransmissão. O protocolo SPEED tem um mecanismo chamado de *backpressure-rerouting* que é usado para lidar com os problemas causados por espaços nas rotas quando um nodo na rota falha ou quando ocorre um congestionamento. Dessa forma, os nodos na rota enviam mensagens de aviso para o nodo fonte a fim de evitar congestionamento, sinalizando ao nodo fonte para procurar novas rotas.

[2] do inglês "*velocity*"

O SPEED busca pelas rotas mais rápidas para atender os requisitos temporais das mensagens, além de prover um mecanismo para evitar o congestionamento das rotas. Entretanto, o protocolo foi projetado para redes esáticas, e o fato de realizar manutenção de rotas em ambiente móvel não é viável devido à constante mudança de topologia. Fazer a manutenção de rota em ambiente móvel tende a causar muito overhead na rede de sensores, além de tornar difícil o cumprimento de prazos na entrega de mensagens, pois seria necessário re-construir as rotas caso os enlaces deixassem de existir.

O MMSPEED (FELEMBAN; EKICI, 2006) é uma derivação do protocolo SPEED. Enquanto o SPEED procura por rotas mais rápidas para entrega de mensagens, o MMSPEED procura também por rotas alternativas às rotas mais rápidas. Dessa forma, o protocolo aumenta a quantidade de rotas para atender aos diferentes tipos de requerimentos temporais. O MMSPEED também foi projetado para redes estáticas, assim pode apresentar os mesmos problemas que o SPEED em ambiente móvel.

Protocolo RPAR

Em (CHIPARA et al., 2006) foi proposto o RPAR (*Real-time Power-Aware Routing*), o qual considera o cumprimento de prazo fim a fim na entrega de mensagens. O RPAR adapta a potência de transmissão de acordo com o prazo do pacote, dessa forma garantindo o cumprimento de requisitos temporais. Com o ajuste da potência de transmissão para valores baixos sempre que possível, os autores alegam que o protocolo pode gastar menos energia dos nodos. Além disso, o protocolo utiliza o controle reativo de vizinhança para manter uma tabela de vizinhos em vez de manter rotas. Assim, a escolha para retransmissão é feita nodo a nodo. Essa característica é interessante para ambiente de mobilidade, pois não é necessário manter rota, somente a vizinhança imediata de um salto, o que torna menos onerosa a manutenção da vizinhança. Entretanto, utilizar mecanismo de controle reativo para manter a tabela de vizinhança atualizada e ao mesmo tempo garantir requisitos temporais pode não ser possível.

Protocolo PATH

O protocolo PATH (REZAYAT et al., 2010) foi proposto na mesma linha que o RPAR. O PATH, entretanto, lida com os requisitos de forma diferente. Primeiramente, o protocolo se preocupa com a energia consumida pelo nodo, computando a energia residual de cada nodo em uma métrica de qualidade de serviço para a escolha do melhor nodo retransmissor. Em segundo, a velocidade do pacote não é obtida pelo ajuste da potência de transmissão, mas pela computação da distância entre o nodo fonte e o nodo *sink*.

No PATH, a manutenção da tabela de vizinhos é realizada com informações na distância de dois saltos de vizinhança. AlÃ©m disso, a escolha do nodo retransmissor depende de métricas que levam em consideração a velocidade da retransmissão que um nodo vizinho pode prover, bem como a sua eficiência energética para retransmitir o pacote. Entretanto, caso não haja nodos elegíveis, o protocolo realiza uma busca reativa por nodos na vizinhança que possam atender aos requisitos. Essa característica pode ocasionar a perda de cumprimento de prazo na entrega de mensagens. Outra desvantagem do protocolo PATH é que não foi considerada a questão de lidar com vários nodos transmissores, por isso não apresenta nenhum mecanismo para lidar com congestionamento.

Protocolo TAEE

O protocolo TAEE (LIU; HONG, 2007) foi projetado para prolongar o tempo de vida da rede, mantendo o conhecimento da energia residual dos nodos e também ficando ciente das condições do tráfego de pacotes. Entretanto, o TAEE assume que a rede seja estática, então a informação da topologia da rede é espalhada somente em tempo de implantação dos nodos. Como se trata de um protocolo de roteamento geográfico, os autores provêem a informação de posicionamento dos nodos manualmente por um GPS. No TAEE, o nodo *sink* utiliza o algoritmo de Floyd-Warshall (CORMEN et al., 2001) para construir a árvore espalhada mínima entre o *sink* e os nodos folhas da rede. Como a informação da árvore construída pelo *sink* é espalhada pela rede, os nodos da árvore proativamente espalham suas condições de energia residual e de tráfego pelos caminhos da árvore, assim essas informações são utilizadas para o roteamento das mensagens.

Apesar das características apresentadas pelo TAEE, que é a preocupação com o consumo de energia e com o congestionamento, o protocolo foi projetado para redes de sensores estáticas. O próprio fato de criar árvores em ambiente móvel deveria ser feito com cuidado, pois a constante mudança de topologia pode causar a desatualização das informações da árvore ou, caso haja uma manutenção a um curto período, pode causar um esgotamento da rede de sensores.

Resumo

A Tabela 5.1 resume os atributos utilizados pelos protocolos descritos para realizar as decisões de roteamento. Como se pode notar, a velocidade é uma preocupação para a maioria dos protocolos que atendem a requisitos temporais. Entretanto, não se observa nenhum protocolo que considere também as seguintes características: controle

de congestionamento, estimação de qualidade de enlace e eficiência energética. A partir disso foi concluído que não havia um protocolo capaz de lidar ao mesmo tempo com mobilidade, alta carga na rede e requisitos temporais. Por essa razão, foi proposto um novo protocolo de roteamento geográfico chamado RACE, conforme descrito na próxima seção.

Tabela 5.1: Atributos cobertos pelos protocolos pesquisados.

Protocolos	Velocidade	Qualidade do *Link*	Congestionamento	Energia
RAP	**X**	-	-	-
SPEED	**X**	-	**X**	-
MMSPEED	**X**	-	**X**	-
TAEE	-	-	**X**	**X**
PATH	**X**	**X**	-	**X**
RPAR	**X**	-	-	**X**

5.2.3 Protocolo RACE

Diante dos desafios apresentados para prover qualidade de serviço em aplicações de RSSF com mobilidade, um novo protocolo de roteamento foi desenvolvido. Esse protocolo foi nomeado RACE (*netwoRk conditions Aware geographiCal forwarding protocol for rEal-time applications in mobile wireless sensor networks*) (ARAÚJO; BECKER, 2011).

O RACE foi projetado para suportar aplicações de nodos sensores móveis que se comuniquem com nodos *sinks* estacionários. A principal característica do protocolo é o provimento de qualidade de serviço relacionada ao tempo de entrega das mensagens.

Essencialmente, o protocolo RACE foi projetado sob a ótica de um algoritmo de roteamento geográfico, na qual os nodos são cientes de suas localizações geográficas. O RACE analisa vários aspectos da rede para realizar as decisões de roteamento, sendo considerado um protocolo multicamadas, visto que as informações utilizadas não se encontram somente na camada de rede, mas também nas camadas de enlace e física.

O RACE mantém uma tabela de vizinhança atualizada a fim de realizar decisões de roteamento de maneira rápida. O roteamento é feito nodo a nodo, sendo que cada nodo calcula uma pontuação para os nodos na sua vizinhança e então escolhe o melhor nodo para a retransmissão das mensagens. Esse mecanismo se mostra adequado para situações onde há mobilidade dos nodos.

O mecanismo de transmissão adotado pelo RACE tem três objetivos principais: (i) entregar mensagens de tempo real dentro do prazo; (ii) entregar mensagens comuns em modo de melhor esforço; (iii) evitar congestionamento local para prover um balanceamento de carga na rede.

Como bem observado em (CHIPARA et al., 2006), situações de congestionamento degradam o desempenho dos protocolos de roteamento, pois mais pacotes são perdidos, aumentando a necessidade de retransmissão de mensagens e, consequentemente, a taxa de ocupação do canal. Por esse motivo, o RACE foi projetado para monitorar esse tipo de situação.

Ele se utiliza de informações relacionadas à quantidade de *buffer* restante (*buffer* residual) como estratégia para evitar a perda de pacotes devido ao congestionamento da rede. Dessa forma, cada nodo no RACE precisa manter o conhecimento cumulativo da situação de *buffer* da vizinhança de um salto até o *sink*.

O RACE também implementa um mecanismo de diferenciação de serviço que prioriza pacotes de tempo real em detrimento dos outros pacotes. As métricas de velocidade e de qualidade de *link* são os principais atributos a serem observados para a escolha do nodo retransmissor quando a mensagem possui requisitos temporais. O uso da velocidade tende a escolher o nodo geograficamente mais perto do nodo *sink*. Entretanto, o uso exclusivo dessa métrica tende a tornar a retransmissão gulosa, o que pode causar congestionamento ao longo do caminho pela escolha viciada dos mesmos nodos. Além disso, o uso da qualidade do *link* para escolha do melhor nodo diminui a probabilidade de retransmissão dos pacotes.

Para uma mensagem cujo objetivo é a entrega pelo melhor esforço, a qualidade do *link* e a situação de *buffer* residual devem ser as principais características a serem observadas para a escolha do melhor nodo. Entretanto, para o uso efetivo das métricas, é necessário um balanceamento apropriado. Por essa razão, o protocolo RACE permite que se estabeleçam os pesos de cada métrica. Isso serve para direcionar a escolha do melhor nodo para retransmissão, i.e., aquele que contém as melhores condições para atender aos requisitos de qualidade de serviço exigidos pela mensagem.

A fórmula usada para se computar a escolha do melhor nodo para se encaminhar uma mensagem é exibida na equação 5.1. O termo $Score_n$ representa a pontuação de cada nodo n existente na tabela de vizinhança. Já o termo LQE representa o resultado de um estimador de qualidade de enlace. Por sua vez, o termo B_r representa o *buffer* residual e finalmente S_p representa a velocidade provida pelo nodo vizinho. O peso de cada métrica é atribuído ao γ_i, cujo $\sum_{i=1}^{3} \gamma_i$ deve ser igual a 1. Cada peso pode ser dinamicamente ou estatisticamente definido, dependendo das necessidades da aplicação.

$$Score_n = \gamma_1 * LQE + \gamma_2 * B_r + \gamma_3 * S_p \quad (5.1)$$

O processo de escolha do melhor nodo para retransmitir a mensagem é resumida no Algoritmo 5.1. A saída do algoritmo é o índice do nodo da tabela de vizinhança com a melhor pontuação. Como entrada no algoritmo tem-se as métricas (velocidade, LQE e *buffer* residual), os pesos e o conjunto de nodos vizinhos.

Algorithm 5.1 Algoritmo para Escolha do Melhor Nodo Retransmissor

1: **Input:** Conjunto de Nodos Vizinhos, Métricas e Pesos das Métricas
2: **Output:** Índice do nodo retransmissor ($fwdNode$)
3: $maxScore = -1$;
4: **for all** Neighbor i **do**
5: $currentScore = weight.lqe * Neighbor[i].lqe + weight.B_r * Neighbor[i].br + weight.S_p * Neighbor[i].sp$;
6: **end for**
7: **if** $maxScore < currentScore$ **then**
8: $maxScore = currentScore$;
9: $fwdNode = i$;
10:
11: **end if**

Leitores interessados em maiores detalhes sobre os componentes internos do RACE podem consultar Araújo e Becker (2011) ou Araújo (2011).

Avaliação de Desempenho do RACE

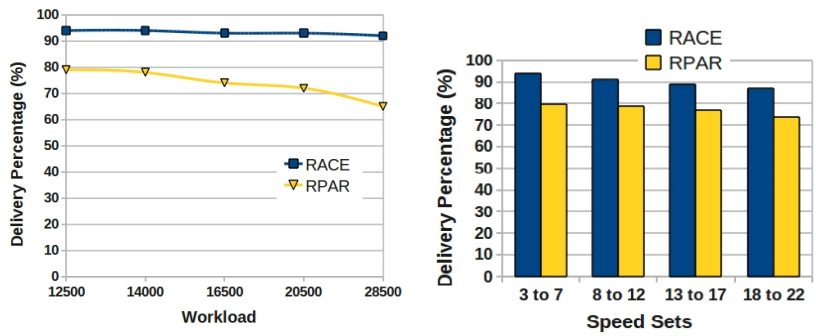
O protocolo RACE foi implementado e avaliado usando a ferramenta de simulação OMNET++ versão 4.1 (VARGA et al., 2001), juntamente com o *framework* INETMANET (VAGAS, 2010). Como camada MAC foi utilizado o IEEE 802.15.4 com CSMA puro, sem as mensagens RTS e CTS. Utilizou-se uma topologia *peer-to-peer*. As configurações da camada física e outros aspectos dos experimentos são mostrados na Tabela 5.2. Para fins de comparação, implementou-se também o protocolo RPAR.

O cenário de simulação escolhido consiste de 300 nodos sensores se movendo ao longo de uma trajetória circular (como uma pista de corrida), com raio de 2.500 metros. Foram colocados 8 *sinks* ao longo do circuito para coletar os dados dos sensores, os quais enviam periodicamente dois tipos de mensagens: mensagens de melhor esforço e mensagens de tempo real.

No caso do protocolo RACE, foram utilizados pesos distintos para cada tipo de mensagens (equação 5.1). Nas mensagens com restrições temporais, os respectivos pesos usados para LQE, *Buffer* e Velocidade de Transmissão foram 0,5; 0,2 e 0,3. Já para as mensagens de melhor esforço os pesos escolhidos foram 0,5; 0,4 e 0,1.

Tabela 5.2: Parâmetros Gerais de Simulação

Parâmetros de Simulação	Valor
Sensitivity	-85 dBm
Default Transmission Power	5 mW
Thermal Noise	-110 dBm
Number of Nodes	300
Number of Sinks	8
Playground Size	2600 x 2600 m^2
Simulation Time	1000 seconds
Mobility Model	Circle
Circle Radius	2500 meters



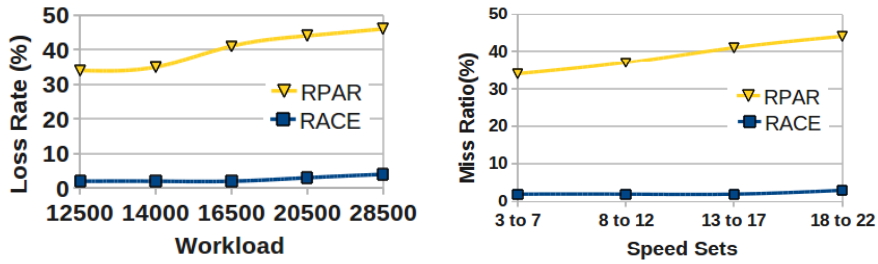
(a) Variação do Workload

(b) Variação da Velocidade

Figura 5.4: Taxa de entrega de dados

Foram realizados dois conjuntos de experimentos para avaliar a taxa de entrega de mensagens. No primeiro conjunto se fixou a velocidade dos nodos para variar a carga de mensagens. Já no segundo conjunto se fixou a taxa de transmissão e se variou a velocidade. Em ambos os casos o RACE teve melhor desempenho em comparação com o RPAR, conforme pode ser observado na Figura 5.4.

Nos experimentos realizados também se computou a taxa de deadlines perdidos (considerando apenas as mensagens de tempo real). O deadline para entrega de mensagens tempo real foi fixado em 100 milisegundos. Novamente o protocolo RACE teve um melhor desempenho em relação ao protocolo RPAR, como mostrado na Figura 5.5. A taxa de perda de prazo do RACE foi no máximo de 4% em ambos os experimentos, com carga de mensagens e velocidades variáveis. Por outro lado, o RPAR foi sensível



(a) Variação do Workload

(b) Variação da Velocidade

Figura 5.5: Taxa de perda de deadlines

a variações de carga de mensagens e velocidade, e teve uma perda média que variou de 34% a 46% no primeiro experimento (variação de carga de mensagem) e de 34% a 44% no segundo experimento (variação da velocidade). Um detalhe interessante é que a média de saltos realizados pelos pacotes para atingir o *sink* foi de 6 saltos, sendo que o máximo foi de 10.

O comportamento do protocolo RACE em relação às taxas de perdas de deadline pode ser explicado pelo uso do mecanismo de *Diferenciação de Serviço*, que prioriza a transmissão de pacotes com prazos associados. Por isso o protocolo RACE consegue sustentar uma taxa linear de perda de deadline em ambos experimentos.

A última questão observada diz respeito ao consumo de energia. Embora isso não seja uma questão fundamental para o cenário de aplicação vislumbrado, é desejável que os protocolos sejam energeticamente eficientes. O resultado obtido para o consumo de energia é mostrado na figura 5.6. Como se pode ver, a média de consumo de energia do protocolo RACE foi próxima à do protocolo RPAR. Detalhe é que este último foi especialmente concebido para ser um protocolo energicamente eficiente. A diferença de consumo entre os dois protocolos foi de 0,5% a favor do protocolo RPAR. O modelo de consumo de energia adotado é baseado em (LANDSIEDEL; WEHRLE; GÖTZ, 2005).

5.3 Controle de Topologia

Dois dos principais aspectos relacionados ao desenvolvimento de protocolos de comunicação em RSSF são eficiência energética e interferência. Protocolos a serem utilizados em RSSF devem minimizar o gasto de energia e a interferência nas trocas de mensagens entre os nós. A minimização do gasto de energia faz com que se prolongue

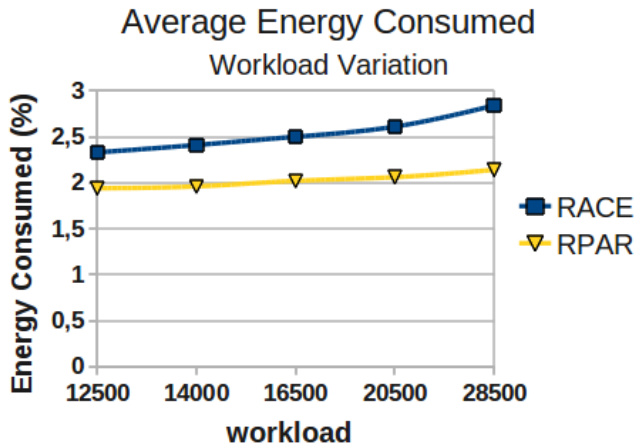


Figura 5.6: Média de Energia Consumida em mW-sec

o tempo de vida da rede. A minimização de interferência diminui colisões (que levam a gasto de energia) e permite melhor utilização espacial.

Eficiência energética e interferência estão diretamente relacionadas à potência utilizada nas transmissões de cada um dos nós. Os transceptores utilizados nos nós sensores tipicamente permitem a transmissão em diferentes níveis de potência. Controlando-se a potência de cada nó permite-se controlar o alcance de cada transmissão. Transmissões a diferentes níveis de potência, realizadas pelos nós, definem diferentes topologias para a rede.

Uma forma, portanto, de se controlar o gasto energético e a interferência da rede é através do controle de transmissão de cada nó, ou seja, através do *Controle de Topologia*. O objetivo de controle de topologia é determinar uma potência de transmissão para cada nó da rede de forma a que uma determinada propriedade seja válida (por exemplo, conectividade) sobre o grafo de comunicação resultante, ao mesmo tempo em que se reduz a energia consumida pelos nós e/ou a interferência (SANTI, 2005).

Na área de controle de topologia, diversos problemas são tratados. Basicamente, os problemas consistem em, a partir de um grafo de conectividade $G = (V, E)$, em que V representa os nós da rede e E os enlaces de comunicação entre os nós, determinar um subgrafo H de G que possua uma certa propriedade. O grafo H deve ser tal que induza uma diminuição no gasto de energia e/ou interferência. A propriedade específica a ser mantida em H determina um problema particular. Exemplos de propriedades são: conectividade (ou conectividade forte); existência de uma árvore de difusão (*broadcast*) com raiz em um determinado nó; *spanners* (energéticos, euclidianos, de saltos);

limitação do grau de um nó, dentre outros. Outros problemas, como Conjunto Independente Maximal (MIS, do inglês *Maximal Independent Set*) e Conjunto Dominante Conexo (CDS, do inglês *Connected Dominating Set*), também têm sido considerados como controle de topologia.

Nesta seção, apresentaremos alguns aspectos relacionados a modelos utilizados em algoritmos de controle de topologia (seção 5.3.1) e dois exemplos de problemas de controle de topologia (seção 5.3.2): conectividade e o problema conhecido como *Subgrafo de Difusão de Consumo Mínimo de Energia* (MECBS, do inglês *Minimum Energy Consumption Broadcast Subgraph*) (CLEMENTI et al., 2001a).

5.3.1 Modelos

Nesta seção são apresentados alguns dos modelos utilizados para representar: conectividade entre nós; gasto de energia; e interferência.

Modelos de Conectividade

Modelos de conectividade definem os relacionamentos de comunicação direta entre nós, ou seja, define quais nós poderão receber uma transmissão de cada um dos demais nós.

A abrangência de comunicação de um nó depende: da sua potência de transmissão; da intensidade com que o sinal atinge o recebedor; e da sensibilidade de recepção do nó recebedor.

Transceptores de nós sensores tipicamente possuem um conjunto limitado de níveis de potência. Transmissões em maior potência resultam em alcances de transmissão mais longos.

Transmissões via canais sem fio estão sujeitas a fenômenos que irão interferir na intensidade que um sinal terá a uma determinada distância do nó transmissor. Três dos principais fenômenos que afetam o sinal são reflexão, refração e espalhamento (*scattering*) (RAPPAPORT, 1996). Em função destes fenômenos, diferentes pontos a uma mesma distância do nó transmissor podem ter valores de potência de sinal distintos, o que faz com que a área (ou volume) de alcance de um sinal não seja regular. Diferentes modelos existem para descrever o decaimento da potência do sinal em função da distância, como, por exemplo, o *free space*, o *log-distance path loss* e o *log-normal shadowing* (RAPPAPORT, 1996).

Um nó conseguirá interpretar o sinal transmitido por um outro nó se a potência do sinal estiver acima de um determinado limiar, chamado de *sensibilidade*, característico

do transceptor do nó recebedor. Na presença de ruídos, a capacidade de um nó receber uma mensagem depende da *relação sinal/ruído*. Um nó consegue interpretar uma mensagem somente se a relação sinal/ruído estiver acima de um limiar.

O modelo de conectividade é representado por um *grafo de conectividade* $G = (V, E)$, em que V representa os nós da rede e E representa as relações de comunicação (alcance) entre os nós. Dois modelos de conectividade bastante comuns são:

1. *Unit Disk Graph* (UDG): modelo mais simples de conexão, modela a situação em que cada nó possui um raio máximo de alcance, normalizado como distância 1. Um nó v será vizinho de um nó u no UDG se e somente se a distância entre u e v for menor ou igual a 1.

2. *Quasi Unit Disk Graph* (qUDG): estende o modelo UDG definindo a conectividade dos nós com base em dois parâmetros: R e r ($R \geq r$). A aresta $(u, v) \in E$ se a distância entre u e v for menor ou igual a r. A aresta $(u, v) \notin E$ se a distância entre u e v for maior que R. A aresta (u, v) poderá pertencer ou não a E se a distância entre u e v for maior que r e menor ou igual a R.

Observe que o UDG é um grafo simétrico, ou seja, se u pode receber uma mensagem de v, então v pode receber uma mensagem de u. O grafo qUDG pode ser assimétrico. Outros modelos foram definidos na literatura, entre eles o *Unit Ball Graph* (UBG) e o *Bounded Independence Graph* (BIG) (SCHMID; WATTENHOFER, 2006).

Modelos de Energia

O modelo de energia define o gasto de energia associado a cada operação realizada pelos nós. Os nós gastam energia na realização de cada uma de suas tarefas, como, por exemplo, durante a transmissão e recebimento de mensagens, processamento de dados e sensoriamento do meio. Os gastos com comunicação são tipicamente considerados os mais relevantes.

A quantidade de energia gasta pelo transceptor de um nó depende do estado operacional em que se encontra. Comumente são definidos os estados de *transmissão* e *recebimento*, além de estados que resultam em menor gasto de energia, como *inativo* (*idle*) ou *dormência* (*sleep*), em que partes do circuito do transceptor estão inativas.

Tipicamente, o gasto de energia associado ao processamento é desconsiderado. Em relação a comunicação, considera-se comumente apenas o custo de transmissão ou o custo de transmissão associado apenas ao custo de recebimento da mensagem por um único nó. Estas suposições resultam em modelos de energia *simétricos*, ou seja, o custo

para se transmitir uma mensagem de um nó u para um nó v é igual ao custo associado à transmissão de uma mensagem do nó v para o nó u.

Alguns trabalhos, no entanto, consideram modelos *assimétricos* de energia. Estes modelos são mais genéricos, uma vez que, além de incluirem os casos simétricos, permitem a modelagem de aspectos adicionais, como: custo de *overhearing*, ou seja, o custo associado ao recebimento, por nós, de mensagens que não são destinadas a eles; e redes heterogêneas, em que diferentes nós podem gastar quantidades distintas de energia para transmitirem a uma mesma potência ou para receberem uma mesma mensagem.

O gasto de energia resultante da comunicação entre um nó u e um nó v é comumente definido como a seguir. Para cada aresta (u, v), $cost(u, v)$ denota a energia gasta por *bit* pela rede quando o nó u transmite com a potência mínima necessária para alcançar o nó v. Assim, $cost(u, v)$ é definido como:

$$cost(u,v) = cf(u) + \gamma(u).d(u,v)^{\alpha} + \sum_{s \in V:(s \neq u) \wedge (d(u,s) \leq d(u,v))} cr(s) \qquad (5.2)$$

onde: $cf(u)$ é a energia (fixa) gasta pelo circuito eletrônico do transmissor no nó u; $\gamma(u)$ é um parâmetro característico do transceptor e do canal sem fio (RAPPAPORT, 1996); $d(u, v)$ é a distância (em metros) entre u e v; α é o componente de perda no canal ($2 \leq \alpha \leq 6$) (RAPPAPORT, 1996); e $cr(s)$ é o custo de recebimento pelo nó s.

Modelos de Interferência

Algoritmos para controle de topologia inicialmente tinham foco apenas em eficiência energética. Interferência era considerada apenas implicitamente, assumindo-se que a redução no número de vizinhos dos nós resultaria em baixa interferência. No entanto, baixo grau de nó não implica necessariamente em baixa interferência (BURKHART et al., 2004). Por isto, interferência deve ser considerada explicitamente. Desde então diferentes enfoques para controle de topologia que procuram otimizar interferência foram propostos. Eles são baseados em métricas que consideram o *número* de nós em áreas específicas afetadas por transmissões. Estas áreas são dependentes da métrica de interferência utilizada.

Pelo nosso conhecimento, Burkhart e outros foram os primeiros a considerarem interferência explicitamente (de uma maneira independente de tráfego), após indicarem que reduzir o grau dos nós não implica em baixa interferência (BURKHART et al., 2004). Os autores utilizaram como métrica de interferência a quantidade de nós presentes no raio de alcance de dois nós comunicantes, o emissor e o recebedor. Estes nós

são os que poderiam escutar a transmissão do emissor e uma resposta correspondente (como uma mensagem ACK, por exemplo) do recebedor. Este enfoque é classificado como *baseado em enlace* (*link-based*), de acordo com a classificação apresentada por Li e outros (2005), porque interferência é baseada nos dois vértices dos enlaces, e como *centrado no emissor* (*sender-centric*), de acordo com a classificação apresentada por Fussen, Wattenhofer e Zollinger (2005), porque interferência é considerada da perspectiva do nó que envia a mensagem.

Mais tarde, Blough e outros (2005) argumentaram que interferência deve ser tratada levando-se em consideração caminhos multisaltos, já que mensagens em geral trafegam através de tais caminhos. Um algoritmo que tenha sido desenvolvido considerando-se apenas caminhos de um único salto não necessariamente fornece uma boa solução quando inteferência de multisaltos for considerada. Os autores assim propõem uma métrica de interferência que considera multisaltos, em um enfoque *baseado em enlace*.

Posteriormente, considerou-se que interferência deveria ser considerada da perspectiva do recebedor, já que uma transmissão pode ser afetada pelo efeito global de diferentes transmissões percebidas pelo recebedor (FUSSEN; WATTENHOFER; ZOLLINGER, 2005; JOHANSSON; CARR-MOTYCKOVÁ, 2005; MOSCIBRODA; WATTENHOFER, 2005; RICKENBACH et al., 2005). Com isto, métricas de interferência classificadas como *centradas no recebedor* (*receiver-centric*) foram criadas. Estas métricas baseiam-se na suposição de que a interferência em um nó v pode ser medida como a quantidade de nós u tais que v está no raio de alcance de transmissão de u. Trabalhos mais recentes consideram interferência baseada em modelos físicos (ao contrário de modelos baseados em grafos, como usado até então) (LIU et al., 2008; GAO; HOU; NGUYEN, 2008). A noção de interferência nos modelos físicos é naturalmente centrada no recebedor.

No entanto, a adoção de um enfoque centrado no emissor captura interferência melhor em certos cenários (DAMIAN; JAVALI, 2008). Adicionalmente, pode haver uma relação entre métricas centradas no recebedor e no emissor (BURKHART, 2003). Foi mostrado que uma métrica centrada no recebedor definida por Rickenbach e outros, chamada I_{in} (RICKENBACH et al., 2005), relaciona-se com uma métrica centrada no emissor definida por Burkhart e outros, chamada I_{out} (BURKHART et al., 2004), pela expressão $I_{in} \leq 5.I_{out}$. Neste caso, reduzir a interferência com uma métrica centrada no emissor implica na redução da interferência centrada no recebedor. Otimizar interferência com base em um enfoque centrado no recebedor é mais difícil, porque o nível de interferência a ser considerado para um nó u depende de possíveis combinações de potências atribuídas a cada um dos nós que pode ter u em seu raio de alcance.

De fato, pelo nosso conhecimento, todos os algoritmos que foram propostos para

otimizar interferência usando um enfoque centrado no recebedor são centralizados, i.e., são baseados em informação global sobre o grafo de comunicação (FUSSEN; WATTENHOFER; ZOLLINGER, 2005; MOSCIBRODA; WATTENHOFER, 2005; RICKENBACH et al., 2005; WU; LIAO, 2008; LIU et al., 2008; GAO; HOU; NGUYEN, 2008). Alguns algoritmos para otimizar interferência baseados no enfoque centrado no emissor são também centralizados: LIFE e LISE (BURKHART et al., 2004) e ATASP (BLOUGH et al., 2005). Entretanto, quatro algoritmos localizados centrados no emissor foram propostos: LLISE (BURKHART et al., 2004), API (JOHANSSON; CARR-MOTYCKOVÁ, 2005), I-LMST (LI et al., 2005) e I-RNG (LI et al., 2005) Um algoritmo distribuído, SLISE (DAMIAN; JAVALI, 2008), também foi descrito, mas otimiza interferência de um salto apenas.

Braga e Assis descreveram um algoritmo de controle de topologia chamado TCO (BRAGA; ASSIS, 2011). Este algoritmo é localizado e baseado em interferência multisaltos, com um enfoque centrado no emissor. De acordo com simulações realizadas, TCO obteve um desempenho superior a todos os algoritmos relacionados da literatura (BRAGA; ASSIS, 2011).

5.3.2 Exemplos de Problemas de Controle de Topologia

Nesta seção, apresentaremos dois exemplos de problemas de controle de topologia: Topologia de Energia Mínima e Subgrafo de Difusão de Consumo Mínimo de Energia.

Topologia de Energia Mínima - *Minimum Energy Topology*

Conectividade é um dos requisitos básicos para o funcionamento de uma rede. As necessidades de conexão, no entanto, podem diferir dependendo da aplicação. Pode-se, por exemplo, desejar que a conexão entre cada par de nós seja simétrica, ou seja, se um nó u pode receber mensagens de um nó v, então v deve também poder receber mensagens de u. Desta forma, pode-se desejar que o grafo que representa a rede seja conexo (quando se assumem grafos não direcionados) ou fortemente conexo (quando se assumem grafos direcionados). Pode-se desejar também que a rede seja k-conexa, ou seja, que existam k caminhos disjuntos (em vértices) entre cada par de nós.

Os problemas de controle de topologia relacionados a estes requisitos são, portanto, determinar a menor potência associada a cada nó de forma a que o requisito seja atendido. Os parâmetros de otimização podem ser distintos, como, por exemplo, minização de grau de cada nó e minimização do gasto de energia total induzido pela topologia.

O problema de se obter um subgrafo conexo ou fortemente conexo com minimização de energia é chamado de *Topologia de Energia Mínima* (CHENG et al., 2003).

Seja $G = (V, E)$ um grafo ponderado. Seja $r : V \rightarrow \mathbb{R}^+$ uma função de atribuição de raio de alcance (*range*) a cada nó de V. O problema Topologia de Energia Mínima consiste em se encontrar um subgrafo $H = (V, E')$ de G ($E' \subseteq E$), induzido por r, que seja conexo (no caso de o grafo G ser não direcionado) ou fortemente conexo (no caso de G ser direcionado) e que minimize $\sum_{v \in V} r(v)$. Este problema é NP-Difícil, tanto para grafos direcionados (CLEMENTI; PENNA; SILVESTRI, 1999) como para não direcionados (CHENG et al., 2003).

Para o caso de grafos não direcionados e pesos de arestas definidas como $t.d(u, v)^\alpha$, em que t é uma constante real, $d(u, v)$ é a distância euclidiana entre dois nós u e v e α é um valor natural, uma definição de potência obtida a partir do cálculo de uma árvore de cobertura de custo mínimo (MST, do inglês *Minimum Spanning Tree*) tem uma razão de aproximação de 2 (CHENG et al., 2003). Ou seja, a soma dos custos associados a uma solução para o problema a partir do cálculo de uma MST é menor ou igual a duas vezes a soma das potências dos nós atribuídas por uma solução ótima.

Subgrafo de Difusão de Consumo Mínimo de Energia -
***Minimum Energy Consumption Broadcast Subgraph* (MECBS)**

Diferentes enfoques para o problema de difusão (*broadcasting*) com eficiência energética em redes sem fio já foram propostos. Este problema consiste em, dado um *nó fonte* s, determinar uma potência de transmissão para cada nó da rede, de modo que: (a) a topologia resultante contenha uma árvore de cobertura com raiz em s; e (b) a soma dos custos associados aos nós seja minimizada.

Este problema foi formulado como o problema *Subgrafo de Difusão de Consumo Mínimo de Energia* (MECBS, do inglês *Minimum Energy Consumption Broadcast Subgraph*) (CLEMENTI et al., 2001a). Seja $G = (V, E)$ um grafo direcionado ponderado com função de custo das arestas $w : E \rightarrow \mathbb{R}^+$. Uma atribuição de alcance (*range assignment*) para G é uma função $r : V \rightarrow \mathbb{R}^+$. O *grafo de transmissão* induzido por G e r é definido como $G_r = (V, E')$, onde:

$$E' = \cup_{v \in V} \{(v, u) : (v, u) \in E \wedge w(v, u) \leq r(v)\}$$

O MECBS é então definido como se segue: dado um *nó fonte* $s \in V$, encontre uma atribuição de alcance r para G tal que G_r contenha uma árvore de cobertura direcionada sobre G com raiz em s e tal que $cost(r) = \sum_{v \in V} r(v)$ seja minimizado.

O MECBS é um problema NP-Difícil (CLEMENTI et al., 2001a) e tem sido extensamente estudado. Entretanto, a maioria dos trabalhos se concentra em versões do problema com custos de arestas simétricos ($w(u, v) = w(v, u)$). Algoritmos centraliza-

dos (e.g., BIP (WIESELTHIER; NGUYEN; EPHREMIDES, 2002)), distribuídos mas não localizados (e.g., (GHOSH, 2008)) e localizados (RTCP e RBOP (CARTIGNY; SIMPLOT-RYL; STOJMENOVIC, 2003), LBOP, LBOP-T e RBOP-T (CARTIGNY et al., 2005), TR-LBOP e TRDS (INGELREST; SIMPLOT-RYL; STOJMENOVIĆ, 2006) e LBIP (INGELREST; SIMPLOT-RYL, 2008)) para versões específicas do problema foram propostos. Em particular, BIP é um algoritmo de aproximação centralizado eficiente (INGELREST; SIMPLOT-RYL, 2008; CALAMONERI et al., 2008).

Algoritmos de aproximação também foram desenvolvidos para o caso de MECBS com custos assimétricos. Algoritmos centralizados foram propostos por Liang (2002), Calinescu e outros (2003) e Ghosh (2008). Os algoritmos apresentados por Calinescu e outros (2003) e Ghosh (2008) fornecem aproximações logarítmicas para o MECBS e melhoram a razão de aproximação do algoritmo apresentado por Liang (2002). O algoritmo apresentado por Calinescu e outros (2003) fornece uma razão de aproximação de $2(ln(n-1)+1)$. O algoritmo apresentado por Ghosh melhora um pouco este resultado, obtendo uma razão de aproximação de $3/2(ln(n-1)+1)$. Não existe algoritmo que ofereça uma razão de aproximação que seja $(1-\epsilon)\ln n$ para qualquer $\epsilon > 0$, a menos que $P = NP$ (n denota o número de nós) (CLEMENTI et al., 2001b; WAN et al., 2002; CLEMENTI et al., 2001a). Portanto, estes algoritmos são assintoticamente ótimos.

Pelo que conhecemos, algoritmos distribuídos que consideram assimetria são somente os apresentados por Li e outros (2006), Ambühl e outros (2004) e Barboza e Assis (2011). Li e outros (2006) apresentam um algoritmo centralizado e um distribuído para construir uma arborescência de difusão (*broadcast*) com atraso de transmissão limitado. O algoritmo não é localizado (é baseado em algoritmos distribuídos para se calcularem caminhos de mínimo custo, árvores de cobertura direcionadas de custo mínimo e busca em profundidade). No algoritmo proposto por Ambühl e outros (2004), os custos das arestas são definidos como o resultado da multiplicação da potência de transmissão pelo *custo de energia unitário* (*energy unit cost*), assim restringindo a forma como os custos das arestas podem ser diferentes. O algoritmo descrito (caso multidimensional, $\alpha > 1$) também não é localizado.

Adicionalmente, Cagalj, Hubauxh e Enz (2002) definiram uma versão do problema com grafos direcionados e custos de arestas potencialmente assimétricos. No entanto, o algoritmo distribuído apresentado é baseado no algoritmo de árvore de cobertura de custo mínimo GHS, que assume grafos não direcionados (CORMEN et al., 2001). Calamoneri e outros (2008) modelam os diferentes níveis de transmissão dos transceptores (em um ambiente homogêneo). Assimetria poderia ser modelada incluindo-se todos os níveis de transmissão de todos os dispositivos de rádio usados em uma determinada

rede. No entanto, o número de tais níveis seria muito grande se considerarmos, por exemplo, assimetria devido a custos de *overhearing* (a restrição de $\Theta(log(n/logn))$ níveis assumida no artigo não se aplicaria no caso geral).

Por fim, Barboza e Assis (2011) descrevem um algoritmo localizado para o MECBS com custos assimétricos, chamado *Localized algorithm for energy-efficient broadcast based on **L**ocal **M**inimum **C**ost **A**rborescences* (LMCA). O LMCA suporta maior flexibilidade na atribuição de custos das arestas do que o algoritmo proposto por Ambühl e outros (2004), em que custos de arestas que saem de um mesmo nó estão associados a um mesmo custo de energia unitário. Barboza e Assis (2011) descrevem também uma variação do LMCA baseada no cálculo localizado de arborescências de caminhos de custo mínimo, chamada de *Localized algorithm for energy-efficient broadcast based on **L**ocal **M**inimum **C**ost **P**aths* (LMCP). Nas simulações apresentadas no artigo, o LMCA obteve melhor desempenho.

CAPÍTULO 6

Desenvolvimento de Sistemas de Controle e Automação na Abordagem de Engenharia Dirigida por Modelos

Leandro Buss Becker, Jean-Marie Farines, Aline Andrade, Ana Patrícia Magalhães

Este capítulo trata sobre metodologias de desenvolvimento de sistemas de controle e automação (SCAs), considerando a hipótese destes sistemas serem embarcados.

SCAs se caracterizam pela reatividade a eventos externos e por apresentarem restrições temporais, de confiabilidade e de segurança. Alguns destes sistemas são críticos em relação à segurança ou ao negócio, o que requer um desenvolvimento rigoroso ao longo de todas as suas etapas, de forma a possibilitar o cumprimento de seus requisitos e a comprovação de sua correção.

A Engenharia Dirigida por Modelos – MDE, *Model Driven Engineering* (SCHMIDT, 2006; KENT, 2002) – se mostra adequada ao desenvolvimento de SCAs, pois garante o rigor do projeto e permite a sua validação ao longo do desenvolvimento. Para tanto, a MDE prevê uma forte ligação entre os modelos projetados e também entre os modelos e o código fonte da aplicação, o qual muitas vezes é gerado automaticamente.

Essencialmente, a MDE é um método de desenvolvimento de sistemas que permite construir uma aplicação, ou parte dela, usando modelos. Diversas Linguagens de Domínio Específico DSLs (*Domain Specific Language*) podem ser utilizadas para representar diferentes aspectos de um sistema durante o seu desenvolvimento, por exemplo, para efeito de projeto, de simulação ou de verificação. O uso de modelos para gerar código ou documentação, validar, simular ou verificar exige a transformação destes modelos de maneira rigorosa. A transformação de modelos é a técnica utilizada para definir mapeamentos e traduções entre diferentes linguagens. A garantia de uma transformação rigorosa é baseada no uso de uma hierarquia de metamodelos para representar linguagens de modelagem. A Seção 6.1 apresenta e discute as principais características da MDE e da transformação de modelos. A Seção 6.2 discute um *framework*, proposto por Magalhaes (2016), para o desenvolvimento de transformações de modelos.

Alguns ambientes (ferramentas) de desenvolvimento de sistemas fornecem mecanismos para a transformação de modelos. Entre eles destaca-se o ambiente TOPCASED (FARAIL et al., 2006), de código aberto e vocacionado para aplicações embarcadas críticas. Um ambiente mais recente e que segue uma linha similar é o Papyrus (PAPYRUS..., 2014). Em relação ao Topocased, ele fornece a base para transformar, de forma rigorosa, modelos escritos numa linguagem de usuário em modelos apropriados para uso com ferramentas de verificação. A complexidade dos SCAs torna necessária a definição de um processo de desenvolvimento no qual se introduzem aspectos da MDE e do ambiente TOPCASED. A seção 6.3 deste capítulo apresenta uma proposta de processo de desenvolvimento de sistemas que possui uma forte ligação com o ambiente TOPCASED.

Finalmente, um aspecto fundamental na garantia do cumprimento dos requisitos é a verificação formal das propriedades do sistema. Os ambientes MDE, em particular o ambiente TOPCASED, contêm cadeias de verificação. Devido à diversidade das linguagens de projeto (e.g. UML, SysML, AADL) e dos formalismos de verificação que estes ambientes devem suportar (e.g. rede de Petri, autômato, álgebra de processo), torna-se vantajoso que estas cadeias de verificação tenham uma linguagem única como passo intermediário da transformação de modelos.

6.1 Engenharia Dirigida por Modelos e Transformação de Modelos

A crescente procura do mercado por software tem impulsionado atividades de investigação e desenvolvimento com o intuito de construir sistemas com alta produtividade.

A Engenharia Dirigida por Modelos (MDE) está inserida neste contexto e objetiva deslocar o foco do desenvolvimento de software, tradicionalmente centrado na programação, para o processo de modelagem do sistema, em que modelos independentes de plataformas são transformados em modelos orientados à implementação (específicos de plataformas) (SELIC, 2003).

A MDE tem potencial para proporcionar um relevante aumento de produtividade na construção de sistemas. A qualidade dos artefatos de software também tem tendência a aumentar, uma vez que são gerados a partir de modelos, construídos com base em metamodelos que especificam conceitos do domínio do problema, e pela utilização de regras de transformação, que facilitam a produção (semi) automática do software. Além disso, os modelos podem ser mapeados em diferentes plataformas permitindo o reuso da modelagem e a consequente redução de esforço de projeto durante o ciclo de vida dos sistemas (YUSUF; CHESSEL; GARDNER, 2006).

Um processo de desenvolvimento de sistemas que utiliza a abordagem MDE precisa estar associado a uma cadeia de transformações que recebe modelos fonte, em um alto nível de abstração, e os transforma em modelos alvo, em níveis mais baixos de abstração, até gerar o código fonte (KURTEV, 2005). A cadeia de transformações de modelos ocupa um papel de destaque na MDE, pois viabiliza, ainda que de forma parcial, a automação do processo.

A Subseção 6.1.1 apresenta os princípios da transformação de modelos. As Subseções 6.1.2 e 6.1.3 detalham respectivamente a especificação e implementação de transformações e aspectos relacionados com a sua correção. A Seção 6.1.4 apresenta as tecnologias que dão suporte à MDE, detalhando um exemplo de abordagem MDE proposta pela OMG, a *Model Driven Architecture* (MDA), e mostra como é tratada a cadeia de transformações nesta abordagem. A Seção 6.1.5 apresenta algumas direções de investigação nesta área. A Seção 6.3 apresenta um framework para construção de transformações de modelos.

6.1.1 Modelos e Transformações

Modelos são especificados a partir de metamodelos. Estes, por sua vez, são também modelos definidos em uma linguagem de modelagem através da restrição e/ou extensão dos construtores da linguagem para representar um espaço de aplicações possíveis dentro de um ou mais domínios. Um metamodelo contém a semântica e a sintaxe necessárias para a construção dos modelos, ou seja, os construtores disponíveis da linguagem de modelagem, e as suas relações e regras de modelagem (STAHL; VOLTER, 2006). Quando um modelo é especificado a partir de um metamodelo ele representa

uma instância daquele metamodelo. Por exemplo, pode-se construir metamodelos para representar o espaço de aplicações dos sistemas de tempo real dirigidos a componentes e a partir destes metamodelos pode-se instanciar modelos para representar sistemas específicos de tempo real dirigidos a componentes. De maneira análoga aos modelos, os metamodelos também podem ser descritos a partir de metametamodelos.

Atualmente existem linguagens de modelagem específicas para um domínio ou uma área de conhecimento, as quais são chamadas *Domain Specific Languages* (DSLs). Também existem linguagens mais gerais, como por exemplo a *Unified Modeling Language* (UML). No caso de uma DSL, ela própria já define um espaço de aplicações, mas metamodelos mais restritos podem ainda ser definidos a partir desta linguagem para representar subespaços de aplicações daquele domínio. Modelos poderiam ser descritos a partir da DSL ou a partir de um metamodelo mais específico. É possível também criar DSLs a partir da UML, que são chamadas de perfis UML. Neste caso, a UML é especializada através de novas restrições que são definidas para atender a um domínio específico, preservando-se a semântica dos elementos já definidos na UML. O perfil MARTE (UML-MARTE, 2008) é um exemplo de perfil UML voltado para o desenvolvimento de SCAs embarcados.

Modelos devem obedecer a uma relação de *conformidade* com seus metamodelos. Um modelo é dito *conforme* com seu metamodelo se estiver sintaticamente correto e atender às restrições impostas pelo metamodelo. Como os modelos são uma especificação de uma aplicação, eles devem satisfazer as propriedades da aplicação, ou seja, devem ser corretos em relação aos requisitos dessa aplicação. Esta relação de correção não é garantida pela relação de conformidade, uma vez que os metamodelos definem propriedades gerais de um conjunto de aplicações e não propriedades específicas de cada aplicação. A garantia de correção dos modelos exige a utilização de técnicas de validação e verificação, como ocorre em outros processos de desenvolvimento de software.

A transformação de modelos consiste numa operação de conversão entre modelo fonte e modelo alvo. Entende-se por modelo fonte aquele que é lido para ser transformado e por modelo alvo aquele que é gerado pela execução da transformação. Na abordagem MDE, modelos são transformados em outros modelos até se chegar ao código fonte da aplicação, através de uma cadeia de transformações.

Uma transformação é definida ao nível dos metamodelos e inclui um conjunto de regras que descreve como modelos fonte podem ser transformados em modelos alvo. Estas regras mapeiam estruturas do metamodelo do modelo fonte em estruturas do metamodelo do modelo alvo.

A Figura 6.1 apresenta os conceitos relacionados com a criação e a execução de uma transformação. O modelo fonte da transformação *mf* está definido em conformidade com o *metamodelo MMf* que, por sua vez, está definido em conformidade com o *metametamodelo MMM*. Como dito anteriormente, as regras que definem a transformação (*Mt*) mapeiam os elementos do metamodelo fonte *MMf* em elementos do metamodelo alvo *MMa*. A execução destas regras é realizada por um *engenho de transformação* que recebe como entrada um modelo fonte *mf* e gera como saída um modelo alvo *ma*, também em conformidade com um metamodelo alvo *MMa*, o qual está também em conformidade com *MMM*.

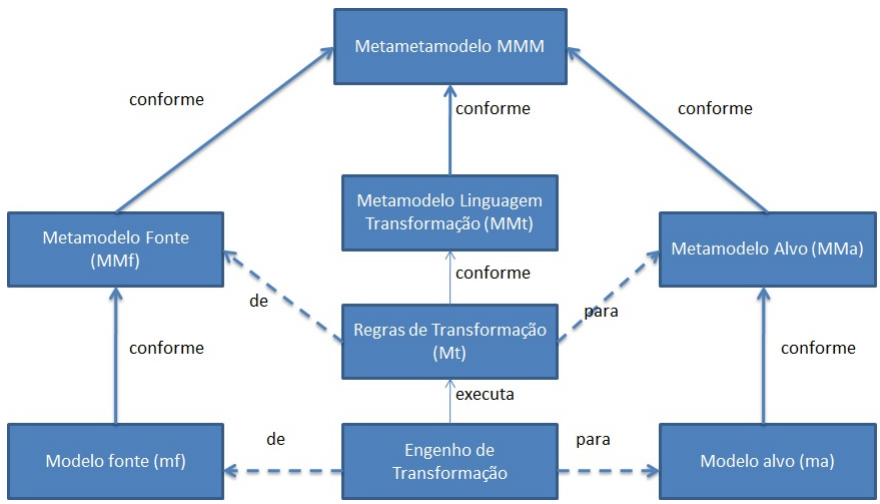


Figura 6.1: Relação entre metamodelo e modelo em uma transformação (adaptado de (BEZIVIN, 2006))

Uma transformação designa-se como *endógena*, se os metamodelos fonte e alvo envolvidos forem o mesmo; e *exógena*, quando os metamodelos fonte e alvo são distintos (MENS et al., 2006). As transformações também podem ser classificadas, quanto ao nível de abstração, em *horizontais* e *verticais*. Na transformação horizontal, os modelos fonte e alvo estão no mesmo nível de abstração e podem estar relacionados, em alguns casos, por uma relação de refinamento. Na transformação vertical, o nível de abstração muda do modelo fonte para o modelo alvo (SENDALL; KOZACZYNSKI, 2003; MENS et al., 2006).

A cadeia de transformações deve preservar a relação de conformidade e de correção dos modelos gerados em cada fase do processo. Desta forma, a geração de código correto depende da construção tanto de modelos como de transformações corretas.

6.1.2 Especificação e Implementação de Transformações

As transformações de modelos são especificadas e implementadas usando *linguagens de transformação* e são apoiadas por ferramentas.

As linguagens de transformação devem prover características que viabilizem: (1) a customização ou reutilização de transformações já existentes; (2) a composição de transformações, para prover maior modularidade e manutenibilidade às transformações; (3) o desenvolvimento de transformações bilaterais, que permitam transformar modelos fonte em alvo e vice versa; (4) o suporte à rastreabilidade, mantendo ligações entre os modelos fonte e alvo que indiquem que elementos do primeiro geraram elementos do segundo; (5) o suporte à propagação de mudanças, provendo alterações incrementais e mecanismos de verificação de consistência; (6) o suporte para validação e verificação dos modelos (MENS et al., 2006).

Transformações podem ser implementadas usando uma linguagem declarativa, imperativa, orientada a objetos ou a combinação destas. A maior distinção entre as linguagens de transformações está nas abordagens declarativa e imperativa. A abordagem declarativa foca em *o que* deve ser transformado *no que*. Abordagens imperativas focam em *como* a transformação deve ser executada (SENDALL; KOZACZYNSKI, 2003; MENS et al., 2006). Existem também abordagens que utilizam regras de transformações de grafos como linguagem para especificar as transformações de modelos (CSERTAN, 2002; VARRO; PATARICZA, 2003).

Transformações se classificam em dois tipos: (1) modelo para modelo (*model2model*) quando um modelo fonte é transformado num modelo alvo; (2) modelo para texto (*model2text*) quando a partir de modelos fonte é gerado como saída o código de implementação do sistema. Existem diversas linguagens para especificação e/ou implementação de transformações *model2model* como, por exemplo, QVT (QVT, 2011) e ATL (ATL, 2011) e *model2text* como, por exemplo, (ECLIPSE, 2005).

6.1.3 Correção de Transformações de Modelos

A correção de uma transformação deve ser considerada sob dois aspectos: sintático e semântico. A correção sintática consiste na preservação da relação de conformidade entre modelos e metamodelos, ou seja, que o modelo Fonte (respectivamente alvo) está em conformidade com o seu metamodelo fonte (respectivamente alvo). Ela deve atender à propriedade de completude, ou seja, a linguagem do metamodelo fonte deve ser completamente coberta pelas regras de transformação, garantindo que as regras incluem todos os elementos da linguagem. A correção semântica consiste na preservação

das propriedades do modelo fonte no modelo alvo após a transformação. A correção sintática não garante necessariamente a correção semântica, uma vez que propriedades referentes aos modelos de aplicações específicas não estão necessariamente especificadas nos metamodelos.

Para garantir qualidade, a transformação deve satisfazer alguns requisitos semânticos, tais como (VARRO; PATARICZA, 2003): (1) terminação, para assegurar que a execução da transformação se conclui; e (2) unicidade, para garantir que a transformação conduz a um mesmo resultado, independentemente da linguagem utilizada.

Em geral, a correção sintática da transformação é facilmente verificada a partir da gramática da linguagem de metamodelagem. A correção semântica, no entanto, é mais complexa, pois exige verificar não só a transformação, mas os modelos envolvidos. Desta forma, classifica-se este tipo de correção em duas abordagens: (1) *off line* quando envolve apenas as regras de transformação e os metamodelos nelas envolvidos; (2) *on line*, quando são analisados os modelos envolvidos na transformação e não a transformação propriamente dita. Considerando que a verificação *off line* é independente dos modelos envolvidos, pode ser feita uma única vez até mesmo de forma manual (ASZTALOS; LENGYEL; LEVENDOVSZKY, 2010).

6.1.4 Tecnologias que apoiam a MDE

O uso da abordagem MDE demanda a utilização de tecnologias que viabilizem a implementação dos conceitos com ela relacionados. A realização mais conhecida da MDE é o framework especificado pela *Object Management Group* (OMG) chamado *Model Driven Architecture* (MDA) (MDA, 2003).

A MDA objetiva aumentar a portabilidade, interoperabilidade e produtividade de sistemas de software, partindo do princípio de que conceitos são mais estáveis que tecnologias, ou seja, o negócio do sistema existe independente de tecnologia. Portanto, é possível modelar o negócio e depois definir em quais tecnologias ele será implementado. A MDA considera a definição de padrões e de uma cadeia de transformações entre três níveis de modelos: (1) modelo independente de computação (CIM – *Computation Independent Model*), (2) modelo independente de plataforma (PIM – *Platform Independent Model*) e (3) modelo específico de plataforma (PSM – *Platform Specific Model*), para então gerar o código fonte do sistema. O CIM é responsável pelo levantamento das necessidades que envolvem o negócio; o PIM é responsável pelo projeto do sistema independente da plataforma; o PSM representa a arquitetura do sistema em uma plataforma tecnológica específica; e o código consiste no programa fonte do sistema (MELLOR, 2004).

Em MDA, modelos são representações abstratas de um sistema e compreendem estrutura e comportamento. Os modelos MDA são construídos normalmente através de perfis UML, que são os mecanismos de extensão padrões desta linguagem. Um perfil adiciona novos elementos ou restrições à UML como, por exemplo, estereótipos e valores rotulados, para representar conceitos de um domínio específico. Desta forma, o desenvolvedor tem à sua disposição uma linguagem com mecanismos de modelagem relativamente alinhados com os conceitos do domínio.

Transformações MDA devem ser desenvolvidas utilizando a linguagem QVT (*Query View Transformation*), padrão OMG para especificação e implementação de transformações. Transformações MDA em geral mapeiam modelos CIM em PIM, PIM em PSM e PSM em código (MOF-Script). Transformações ainda são possiveis de PIM para PIM, PSM para PSM e código para código, em caso de refinamentos. Todas estas transformações são auxiliadas por ferramentas que as executam.

Uma transformação MDA se baseia nos construtores dos perfis UML utilizados em cada nível de abstração. Por exemplo, transformar um modelo PIM para um modelo PSM envolve executar as regras que mapeiam os elementos do metamodelo do perfil PIM para os elementos do metamodelo do perfil PSM.

No contexto da MDA, os modelos UML construídos são representados sob a forma de um arquivo XMI. Assim, os arquivos XMI, que representam modelos e seus respectivos metamodelos, e as regras de transformações são as entradas de uma ferramenta de transformação. Uma vez processada a transformação, é gerado o modelo alvo (representado também por um arquivo XMI) em conformidade com o metamodelo alvo.

6.1.5 Considerações sobre pesquisas em torno da MDE

A MDE é uma abordagem recente e tem sido objeto de vários projetos de investigação com um amplo campo de pesquisas. De uma forma geral, podem-se categorizar estas pesquisas em duas direções: (1) uma com foco na **modelagem de sistemas**, abstraindo a construção da transformação, que envolve metamodelagem, linguagens específicas de domínio e uso de formalismos para a especificação e verificação dos modelos; e (2) outra que trata da cadeia de **transformação de modelos**, envolvendo linguagens para especificação e desenvolvimento da transformação, verificação formal da transformação e ferramentas para a construção e execução da cadeia de transformações. Em seguida, discute-se cada uma destas duas categorias.

Modelagem de sistemas

Tendo como foco a modelagem de sistemas, a principal mudança trazida pela abordagem MDE está no fato dos modelos deixarem de ser vistos apenas como documentação do sistema e passarem a ser considerados como um dos artefatos principais do processo de desenvolvimento que pode inclusivamente ser utilizado para gerar o código da aplicação. Por este motivo uma maior atenção deve ser dada ao nível de detalhe dos modelos. Quanto mais detalhado o modelo, mais completo poderá ser o código gerado a partir deste modelo e menor será a interferência humana no processo de geração desse código. No entanto, modelar aplicações com uma riqueza de detalhes tal que permita gerar de maneira automática o sistema não é uma tarefa fácil, nem sequer desejável do ponto de vista prático.

Linguagens de modelagem de propósito geral como, por exemplo, a UML são muito genéricas e não modelam aspectos específicos de certos domínios. Neste sentido, é consensual a necessidade de utilização de DSLs e, no caso da MDA, de perfis UML, que possam enriquecer a modelagem com conceitos relacionados com domínios específicos. Muitas linguagens foram propostas nos últimos anos para facilitar esta modelagem, a exemplo de perfis para sistemas de tempo real, para processos de negócio, para aplicações web, entre outros (MDA, 2003). Também tem sido utilizada a integração de linguagens, por exemplo, UML com OCL, para adicionar restrições aos modelos.

As linguagens de modelagem são utilizadas na especificação dos modelos estruturais e comportamentais de um sistema. Observa-se atualmente que a modelagem estrutural é um problema relativamente bem resolvido e que a geração de código fonte nestes casos pode ser vista como uma tradução direta dos modelos para as estruturas que irão compor o *"esqueleto"* da aplicação. Uma vez gerada esta estrutura, o programador pode completar o código adicionando o comportamento necessário. No caso de UML, por exemplo, é comum os desenvolvedores utilizarem o diagrama de classes para representar a estrutura de um sistema e a geração do código fonte a partir da estrutura básica destas classes é uma tarefa simples já realizada por diversas ferramentas.

Em relação à modelagem do comportamento do sistema, a tarefa deixa de ser simples por diversos motivos como, por exemplo, a dificuldade em se manter a consistência entre os diversos modelos, comportamentais e estruturais, que constituem a especificação de um sistema. A UML, por exemplo, possui diversos tipos de modelos, cada um com um propósito específico. A integração (1) entre modelos comportamentais e (2) entre modelos estruturais e comportamentais é um desafio para a geração de código. Neste sentido, vários estudos como por exemplo (RICCOBENE; SCANDURRA,

2009) têm sido desenvolvidos para definir uma relação entre os diversos diagramas oferecidos pela UML de forma que eles possam ser integrados na geração de código.

Em relação à correção dos modelos, algumas propostas utilizam linguagens formais para representar o comportamento, outras mapeiam modelos não formais em modelos formais. Por exemplo, máquinas de estados de UML podem ser mapeadas em autômatos para serem processados por verificadores de modelos (MUNIZ; ANDRADE; LIMA, 2010). A especificação de comportamento utilizando modelos formais é tratada em vários trabalhos, particularmente os verificadores de modelos (CLARK; YUILLE, 1990) têm sido muito aplicados, como exemplo o trabalho de (MCNEILE; ROUBTSOVA, 2009) utiliza *Abstract State Machines* e verificadores de modelos para analisar a correção da especificação.

Transformações de modelos

Como dito anteriormente, a cadeia de transformações é crucial para a automação do processo de desenvolvimento na MDE e possui um amplo campo de pesquisa, voltado para a especificação de transformações e a construção de transformações corretas.

A construção de transformações durante muitos anos foi realizada de forma *ad hoc*: especificada em linguagem natural e implementada diretamente no código. (BEZIVIN, 2006) introduziu o conceito de modelo de transformação de modelo, de forma que a transformação pudesse ser especificada em alto nível de abstração antes de ser codificada. Esta idéia desencadeou várias pesquisas, como, por exemplo, as transformações de alta ordem que utilizam transformações como entrada de dados para gerar outras transformações através de composições e uso de padrões. (TISI; CABOT; JOUAULT, 2010). Nesta mesma linha, a OMG propôs a QVT como uma linguagem para especificação e desenvolvimento de transformações. A QVT possibilita a especificação da transformação em alto nível de abstração, a composição de transformações, o uso de padrões de transformação, entre outros aspectos envolvidos na construção de transformações. No entanto, investigações conceituais neste domínio de problema, independentemente do espaço tecnológico, continuam sendo objetos de pesquisa (CHRISTIANO; MENEZES; COMICIO, 2011).

A garantia da correção da transformação também é um campo que atrai muitas pesquisas, devido aos desafios que este problema encerra. Neste tema, as pesquisas estão relacionadas principalmente com a correção semântica das transformações, com o objetivo de garantir que as propriedades de um modelo fonte são preservadas no modelo alvo após a execução de uma transformação. Alguns trabalhos, para viabilizar a verificação de propriedades nos modelos, enriquecem as linguagens com formalismos,

a exemplo de (LANO K.; CLARK, 2008) que utiliza UML com OCL, e de (KIM; BURGER; CARRINGTON, 2005) que utiliza UML com Object Z. Outras abordagens utilizam regras de transformação de grafos para gerar as transformações combinadas com máquinas de estados para verificar propriedades (VARRO; PATARICZA, 2003; CSERTAN, 2002).

6.2 Framework para a construção de transformações de modelos

Transformações de modelos têm sido consideradas o núcleo do desenvolvimento na abordagem MDE, pois encapsulam as estratégias utilizadas para transformar modelos fonte em modelos alvo, que geram artefatos em diferentes níveis de abstração, e possibilitam a (semi) automação do processo de desenvolvimento de aplicações. No entanto, transformações têm sido construídas muitas vezes de maneira *ad-hoc*. Essa conduta dificulta a adoção de boas práticas de desenvolvimento (ex. reuso e padrões) e gera uma documentação deficiente que pode impactar na manutenção e na evolução destas transformações.

Assim, as transformações de modelos são vistas como aplicações que recebem modelos de entrada e produzem modelos de saída. Desta forma, como qualquer outra aplicação, as transformações devem ter um ciclo de vida de desenvolvimento que abranja desde a especificação até a codificação. Sendo assim, é possível especificar uma transformação de modelos, em alto nível de abstração, através de modelos de transformações de modelos, como proposto por (BEZIVIN, 2006). Um modelo de transformação de modelos é construído em conformidade com o metamodelo da linguagem de transformação. Neste caso, as regras de transformações entre os modelos fonte e alvo são instanciadas a partir dos modelos de transformação, ao invés de serem codificadas diretamente na linguagem de transformação. Essa ideia possibilita o uso da própria abordagem MDE para a construção de transformações.

Neste contexto, a construção de modelos de transformação de modelos demanda a utilização de um processo de desenvolvimento para guiar os desenvolvedores na construção dos modelos ao longo de todo o seu ciclo de vida de desenvolvimento, apoiado por linguagens de modelagem apropriadas para o domínio de transformações.

Nesta seção, é apresentado o *framework Model Driven Transformation Development* (MDTD) (MAGALHAES; ANDRADE; MACIEL, 2016), uma solução integrada para desenvolvimento de transformações unidirecionais usando a abordagem de desenvolvimento dirigido a modelos.

O *framework* MDTD compreende os seguintes elementos: (1) linguagem de modelagem independente de plataforma, para construção dos modelos de transformação de modelos; (2) linguagens de modelagem para plataformas específicas, para possibilitar a posterior geração de código em plataformas específicas; (3) processo de desenvolvimento de transformações, para guiar o desenvolvedor; (4) cadeia de transformações, para automatizar o desenvolvimento; e (5) ambiente de desenvolvimento. A Figura 6.2 apresenta uma visão geral destes elementos e suas relações.

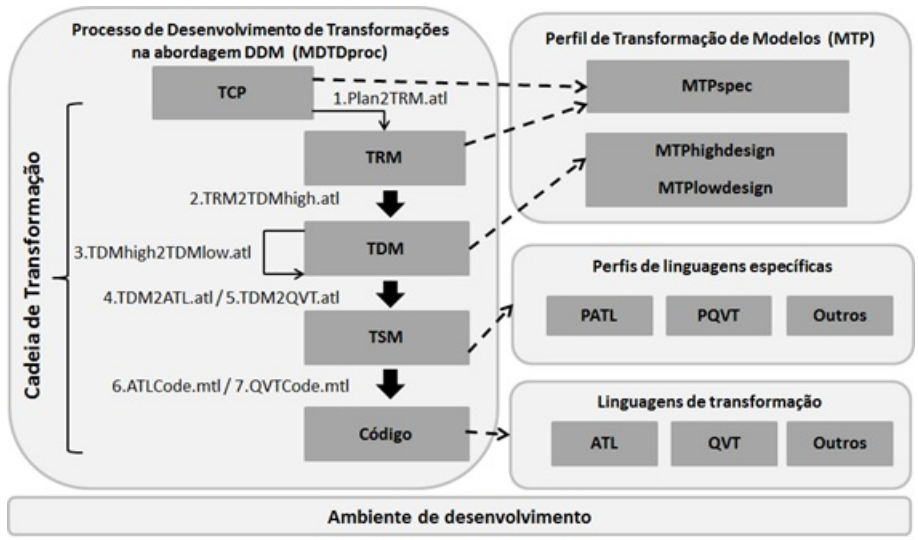


Figura 6.2: Visão geral do *framework*

O processo de desenvolvimento, chamado MDTDproc (lado esquerdo da Figura 6.2) é um processo iterativo e incremental, especificado segundo o metamodelo SPEM, responsável por guiar todo o desenvolvimento através de um ciclo de vida que compreende cinco fases: Planejamento (TCP – *Transformation Chain Planning*), relacionada ao planejamento da cadeia e suas transformações; Especificação (TRM – *Transformation Requirements Modeling*), relacionado ao detalhamento dos requisitos da transformação e a definição dos metamodelos envolvidos; Projeto independente (TDM – *Transformation Design Modeling*), relacionado ao projeto da transformação independente de plataforma; Projeto específico (TSM – *Transformation Specific Modeling*), relacionado ao projeto da transformação em uma plataforma específica; e Código, relacionado ao código da transformação em uma linguagem específica. A construção de transformações não é uma tarefa simples, pois requer expertise em linguagens de transformações e metamodelagem, conhecimentos que não fazem parte do processo tradi-

cional de desenvolvimento de sistemas. O processo proposto conduz o desenvolvedor passo a passo nas várias fases do ciclo de vida de desenvolvimento da transformação, com atividades específicas para este domínio.

A linguagem de modelagem independente de plataforma, chamada Perfil de Transformação de Modelos (MTP – *Model Transformation Profile*), compreende um conjunto de metamodelos e perfis que representam os conceitos do domínio de transformação de modelos. O MTP (no topo à direita da Figura 6.2) apoia as atividades executadas nas três primeiras fases do processo de desenvolvimento de transformações (TCP, TRM e TDM) e permite a modelagem de transformações em diferentes níveis de abstração, independente de linguagem de programação, utilizando diagramas da UML. Desta forma, possibilita o projeto de soluções desassociadas de detalhes específicos de implementação, portáveis para linguagens específicas de transformação. As linguagens de modelagem específicas de plataforma também são definidas sob a forma de perfis UML. Dois perfis compõem o *framework* o perfil PATL (para a linguagem ATL) e o perfil PQVT (para a linguagem PQVT), do lado direito da Figura 6.2. Estes perfis apoiam as atividades do processo de desenvolvimento de transformações na fase de projeto específico de plataforma (TSM), e possibilitam a geração e representação de modelos de transformação nas linguagem ATL e QVT (na base direita da Figura 6.2) usando o diagrama de classes da UML.

A utilização de um perfil para a construção de transformações é relevante porque a UML é uma linguagem padrão de modelagem tanto na indústria quanto na academia. Desta forma sua adoção na construção de transformações além de proporcionar o uso de boas práticas de desenvolvimento e gerar uma documentação consistente (antes embutida no código), pode diminuir a complexidade do desenvolvimento em geral realizado diretamente nas linguagens de programação voltadas a transformações de modelos. Além disso, a UML possui vasto suporte ferramental para modelagem.

A cadeia de transformação, lado direito da Figura 6.2, (semi) automatiza o processo de desenvolvimento através de um conjunto de sete transformações que geram modelos de transformações de modelos em diversos níveis de abstração, desde os requisitos até a geração do código fonte.

O ambiente de desenvolvimento, apresentado na base da Figura 6.2, integra ferramentas de modelagem, ferramentas de metamodelagem e engenhos de transformações, no ambiente Eclipse. Este ambiente utiliza apenas ferramentas livres que podem ser substituídas por outras de mesmo fim. Por exemplo, o uso de um perfil UML para modelagem da transformação possibilita trocar o ambiente de modelagem atual por outra ferramenta de modelagem que utilize a linguagem UML.

A integração do *framework* a novas linguagens de transformação pode ser realizada através da construção de transformações que convertam o modelo independente de plataforma em modelos específicos de outras linguagens.

O *framework* MDTD objetiva aumentar a qualidade das transformações construídas e facilitar o desenvolvimento de uma transformação. Com o *framework* é possível: (1) sistematizar as atividades necessárias para desenvolver as transformações; (2) adotar uma notação padrão para o desenvolvimento de transformações; (3) modelar a transformação em um nível de abstração mais alto, em vez de focar o desenvolvimento diretamente no código; (4) construir transformações bem formadas, que satisfaçam as propriedades importantes para uma transformação, a exemplo de completude; (5) antecipar erros de concepção para as fases de especificação da transformação; (6) permitir a portabilidade.

6.3 Processo de Desenvolvimento de Sistemas de Controle e Automação

Esta seção visa apresentar uma proposta de um processo de desenvolvimento de Sistemas de Controle e Automação (SCAs), em particular quando estes sistemas apresentam características de criticidade, i.e. quando a ocorrência de falhas pode provocar sérios danos, quer ao sistema quer aos seus usuários.

Sabe-se, no entanto, que prover garantias de ausência de falhas não é uma questão trivial, visto que muitos Sistemas de Controle e Automação apresentam elevada complexidade de desenvolvimento e análise. Por isso, o projeto destes sistemas necessita de metodologias e ferramentas que proporcionem a representação clara e precisa de seus elementos em um alto nível de abstração, refinando-os na medida da evolução do processo de desenvolvimento.

O processo de desenvolvimento proposto em (BECKER et al., 2010) pretende garantir o desenvolvimento seguro de SCAs. Entende-se por desenvolvimento seguro quando o sistema resultante passa por diversas etapas de verificação formal de propriedades, capazes de garantir a sua correção. Para alcançar este objetivo, são realizadas várias transformações de modelos que permitem por exemplo a obtenção de um autômato, que representa o comportamento do sistema, capaz de ser verificado formalmente.

A Figura 6.3 apresenta uma visão geral dos passos previstos para o processo de desenvolvimento proposto por Becker e outros (2010).

Seguindo a proposta apresentada na grande maioria dos processos de desenvolvi-

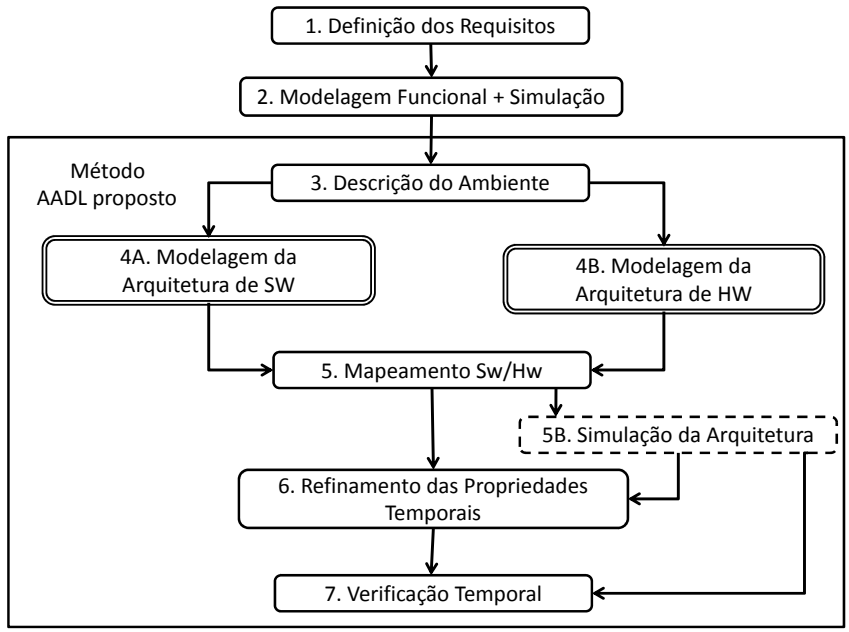


Figura 6.3: Fluxo do processo de desenvolvimento proposto (BECKER et al., 2010)

mento, Becker e outros (2010) sugerem que o primeiro passo seja a especificação dos requisitos da aplicação. Neste passo, é feita a especificação (requisitos funcionais e não funcionais).

No passo seguinte, é desenvolvido o modelo funcional do sistema que representa tanto as características do sistema físico a controlar (também referido como ambiente) como as do controlador propriamente dito. O Matlab/Simulink (SIMULINK, 2011) e o SCADE/Lustre (ESTEREL-TECHNOLOGIES, 2011) são exemplos de ferramentas comumente usadas para tais fins.

Em seguida (passo 3), é gerado um modelo já na linguagem do projeto, por exemplo, em AADL (SAE, 2006), relacionando o sistema e os dispositivos externos com o qual ele interage.

Posteriormente, é feito o projeto da arquitetura do software capaz de satisfazer os requisitos do sistema em questão (passo 4A). Tal arquitetura deve estar em conformidade com o hardware disponível para sua implementação (passo 4B). A especificação

do hardware pode ser realizada em conjunto ou em separado com a da arquitetura do software.

O próximo passo consiste em ligar através de um mapeamento (passo 5) as arquiteturas de software e de hardware. Neste momento, o modelo só não está totalmente completo porque faltam informações a respeito dos tempos de execução das subrotinas de software (passo 6). Tais informações devem ser obtidas através de ferramentas específicas. Em seguida é feita a verificação das propriedades temporais para analisar se o sistema cumpre corretamente com as restrições de tempo previstas.

Durante todo este processo, análises, simulações e verificações permitem testar o sistema em desenvolvimento. Finalizados tais testes, é possível prosseguir com a geração (eventualmente automática) do código da aplicação final.

É importante citar que o fluxo de projeto entre as diversas fases previstas não é necessariamente sequencial. A necessidade de corrigir os problemas detectados pelo processo de verificação pode levar o projetista a retroceder nos passos realizados. Por exemplo, caso ocorram erros no processo de verificação temporal (passo 7), o projetista deve avaliar se o problema é resultante do passo 4 (proposta da arquitetura de software) ou se é do passo 5 (mapeamento entre os recursos de software e de hardware).

Com o objetivo de descrever mais detalhadamente cada um dos passos do processo, estes são ilustrados através de um exemplo simples de um sistema embarcado responsável por realizar manobras de estacionamento automático em veículos (SEAV). Considera-se a linguagem de descrição de arquitetura AADL como a linguagem utilizada pelo projetista.

6.3.1 Definição de Requisitos

O primeiro passo em qualquer metodologia de desenvolvimento é a definição dos requisitos do sistema. Isto inclui tanto as funcionalidades, conhecidas por Requisitos Funcionais (FR, *Functional Requeriments*), quanto as restrições do sistema, chamados Requisitos Não-Funcionais (NFR, *Non-Functional Requirements*) ou de Qualidade.

Para ilustrar, os requisitos relacionados com o SEAV são apresentados na Tabela 6.1. As três principais funcionalidades desse sistema são as seguintes:

FR1. Ligar/desligar o sistema;

FR2. Localizar uma vaga de estacionamento; e

FR3. Estacionar o veículo.

Associados a estes requisitos funcionais, encontram-se subfuncionalidades e eventuais restrições – isto é, requisitos não funcionais, os quais são identificados pela sigla NFR.

Tabela 6.1: Requisitos do SEAV

FR1 - Ligar/desligar o sistema	
Descrição: O motorista do veículo deve ativar o SEAV para que este procura uma vaga.	
FR1.1 - Velocidade máxima	Para iniciar o sistema a velocidade não deve ser superior a 20Km/h.
FR1.2 - Em operação / Terminado	O motorista é informado do estado de operação do SEAV.
FR2 - Localizar vaga	
Descrição: O sistema deve iniciar a localização de uma nova vaga de estacionamento.	
FR2.1 - Alerta ao motorista	O motorista é informado quando uma vaga de estacionamento for localizada.
FR2.2 - Segurança	Se a velocidade exceder 20km/h, então deverá ser emitido um sinal de alerta ao controlador.
NRF2.2.1 - Tempo-Real	O sinal de alerta deve ser emitido em no máximo 10ms.
FR3 - Estacionar o veículo	
Descrição: O motorista deve iniciar o estacionamento após ter encontrada uma vaga. O sistema controla a velocidade e a direção do veículo.	
FR3.1 - Segurança	O sistema pode começar a manobra de estacionamento somente se a velocidade for zero.
FR3.2 - Sinal de Término	O sistema deve alertar o motorista quando a manobra de estacionamento for finalizada.
FR3.3 - Parada de Emergência	O sistema deve parar se o motorista mover a direção.
NFR3.3.1 - Tempo-Real	O sistema deve parar em no máximo 2 segundos.

6.3.2 Modelagem Funcional e Simulação

Muitos SCAs, especialmente os críticos, possuem uma ligação forte com algum sistema físico que precisa ser controlado (automóveis, aeronaves, satélites, equipamentos industriais, etc). Nesse tipo de aplicação, é comum que antes do desenvolvimento do sistema embarcado propriamente dito seja feito o desenvolvimento e ajuste dos algoritmos de controle, levando em conta este sistema físico. Na maioria dos casos, isso é feito por simulação, através de ferramentas como *Matlab/Simulink* ou *Scade/Lustre*. Desta forma, tais simulações servirão como base para o restante do desenvolvimento do SCA.

A Figura 6.4 apresenta em Simulink o nível mais alto da simulação funcional do SEAV.

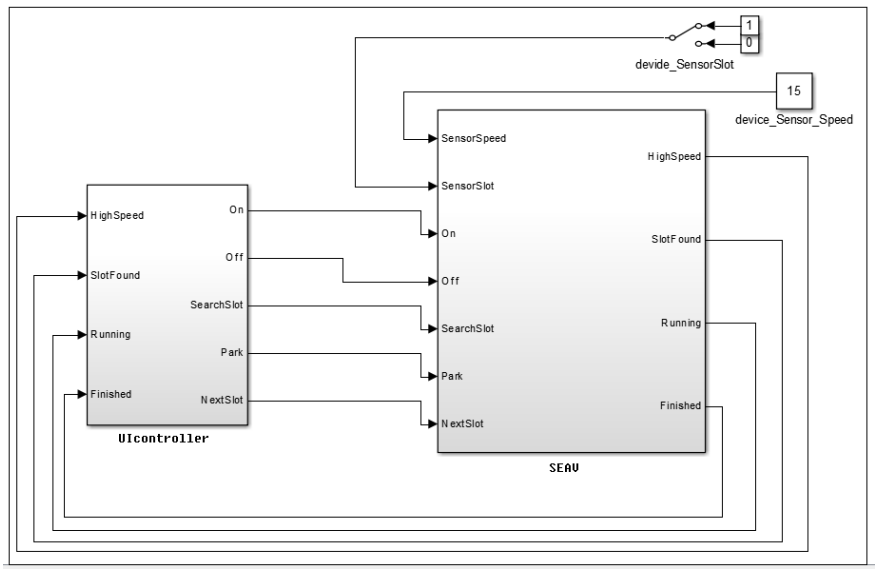


Figura 6.4: Modelo Simulink do SEAV

6.3.3 Descrição do Ambiente

Este passo visa representar o SCA e os dispositivos com os quais ele interage, coletivamente designados como o ambiente. A principal dificuldade encontrada nesse passo é a maneira como as informações são transferidas do modelo funcional para um modelo na linguagem AADL, adotada para representar a arquitetura do sistema.

Num primeiro momento essa transferência é realizada pelo projetista, sem nenhum tipo de apoio de ferramentas computacionais. Entretanto, atualmente já se encontra em andamento um trabalho que visa encontrar alternativas para automatizar esta etapa, através de uma metodologia MDE e do uso de transformação de modelos (PASSARINI; FARINES; BECKER, 2012).

Outro aspecto acerca deste novo modelo a ser gerado, é que ele requer informações que normalmente não fazem parte de um modelo funcional. Por exemplo, é possível detalhar a taxa de chegada das mensagens, seus tipos de dados e suas restrições temporais. Essas informações devem ser detalhadas pelo projetista para permitir uma análise correta da arquitetura em desenvolvimento.

Apresenta-se na Figura 6.5 o diagrama de contexto do SEAV especificado na linguagem AADL, correspondente à simulação em Simulink do SEAV (Figura 6.4). O elemento central, `SEAV`, representa o sistema embarcado propriamente dito. Os elementos ao redor representam os periféricos com os quais ele interage.

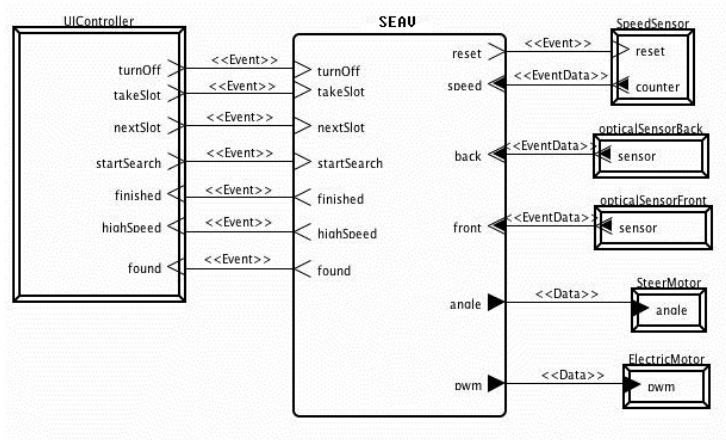


Figura 6.5: Diagrama de Contexto do SEAV

Em relação à representação dos dispositivos externos, supõe-se a existência de dois tipos de dispositivos: (i) dispositivos novos e (ii) dispositivos reaproveitados de outras aplicações. Enquanto dispositivos como sensores e atuadores, a exemplo de SpeedSensor, opticalSensorBack, opticalSensorFront, SteerMotor e ElectricMotor, são normalmente reutilizados de outros modelos existentes, as interfaces com o usuário, como UIController, podem ser consideradas como novos dispositivos para os quais deve-se especificar o comportamento (por exemplo, em autômatos). Estes modelos de dispositivos novos devem passar pelo processo de verificação antes do seu uso no modelo global.

A Figura 6.6 ilustra o autômato de um possível comportamento para a interface `UIController`. É possível verificar formalmente que o usuário sempre consegue parar o veículo, independentemente do estado de execução do sistema no qual se encontra, a partir do evento `Off!` gerado pelo motorista. Este comportamento pode ser verificado formalmente nesta etapa do desenvolvimento, selecionando a interface com o motorista (UIController) e analizando a conformidade de seu comportamento com o desejado, descrito pelo autômato da Figura 6.6. Apesar de sua simplicidade, este exemplo ilustra a possibilidade do uso de verificação formal nesta etapa de desenvolvimento.

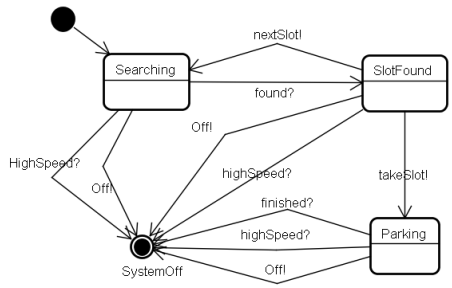


Figura 6.6: Comportamento da Interface com o Usuário - UIController

6.3.4 Modelagem da Arquitetura do Software

Este passo pode ser considerado como o principal dentro do método proposto. Ele terá diversas iterações, uma vez que o projetista poderá refinar o modelo especificado em AADL tantas vezes quanto necessário, até que o nível desejável de detalhe seja alcançado. Cada sub-modelo gerado deverá passar pelas etapas de verificação. Em paralelo a este passo deve ser feita a modelagem da arquitetura de hardware onde o sistema deverá ser executado.

Na primeira iteração o projetista deve detalhar o processo *System* (no nosso caso, `SEAV` - vide Figura 6.5) num conjunto de subcomponentes, que podem ser tanto processos como *threads*. Submete-se então o modelo à verificação formal e, caso o resultado seja falso, um novo refinamento deverá ser realizado. De acordo com essa perspectiva, cada componente do modelo pode derivar diversos subcomponentes. Por definição, o refinamento só termina quando houver informação suficiente para a correção do modelo ser provada pelo processo de verificação.

Cada modelo detalhado (i.e., cada iteração) deve, no entanto, estar conforme com o comportamento abstrato definido pelo componente de mais alto nível. Na Seção 6.4 discute-se a cadeia de verificação formal utilizada.

Ao subdividirmos o sistema, é necessário definir as informações trocadas entre os subcomponentes (ou *threads*). Para tanto, especificam-se portas e os tipos de dados a elas associados.

A decomposição do *System* `SEAV` é exibida na Figura 6.7. *SystemManagement* é usado para ligar ou desligar o sistema através da interface com o usuário (FR1) (Tabela 6.1). Já *SlotSelection* serve para coordenar a localização de uma vaga de estacionamento (FR2) (Tabela 6.1). Finalmente, *ParkManeuver* é responsável por realizar o estacionamento (FR3) (Tabela 6.1).

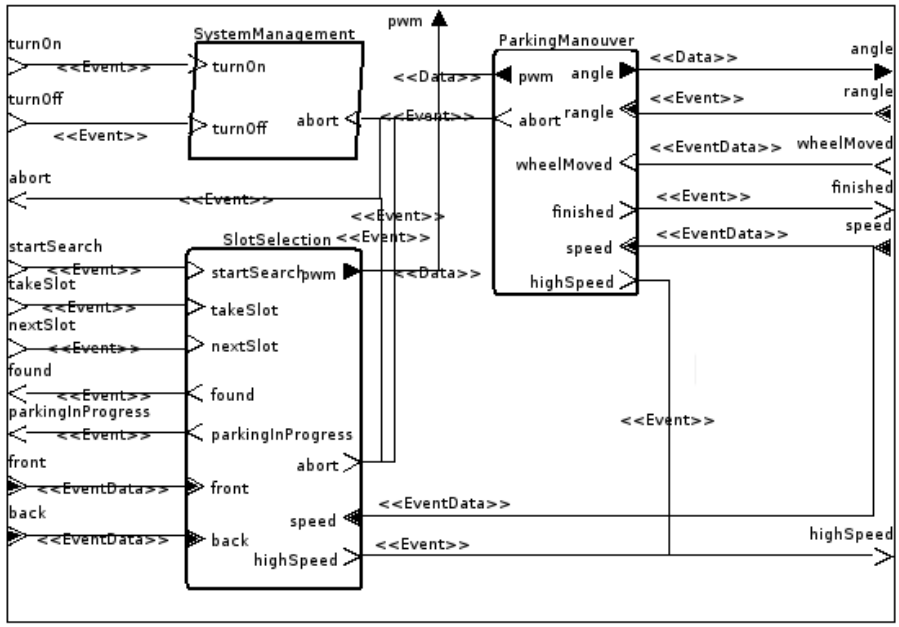


Figura 6.7: Modelo AADL do primeiro nível de decomposição do *System* SEAV

Logo após a decomposição é necessário definir em quais modos de operação (*Slot*, *Manouver*) cada *thread* permanecerá ativa. É possível que uma *thread* esteja associada a diversos modos de operação, sendo que em AADL tal representação é feita diretamente no código.

A ideia de criar modos de operação pode ser vista como uma espécie de decomposição temporal do modelo, considerando seu conjunto de funcionalidades. No exemplo aqui apresentado, os modos de operação (*Slot*, *Manouver*) são análogos aos subcomponentes do primeiro nível de decomposição do modelo AADL (*Slot*Selection, Parking*Manouver*). A Figura 6.8 exibe o autômato que representa o comportamento do sistema SEAV. O uso do Anexo Comportamental da AADL[1] é uma alternativa que permite descrever, neste e em outros níveis de refinamento, o comportamento dos subcomponentes, na forma de um conjunto de transições de estados organizados como um sistema de transições (FRANCA et al., 2007).

[1]Modelo de autômatos para descrever comportamento em AADL.

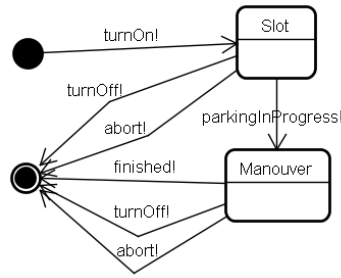


Figura 6.8: Modos de Operação Básicos do *System* SEAV

6.3.5 Etapas Subsequentes

Considera-se que as informações temporais associadas ao modelo AADL sejam apenas estimativas. Para refinar estas informações é necessário a obtenção de informações temporais mais precisas, o que implica na alocação (também chamada *deployment*) dos processos e das *threads* a elementos de processamento (etapa 5 do processo proposto).

Feita esta alocação, supõe-se que o projetista conseguirá informações temporais mais precisas para atualizar o modelo (etapa 6 do processo proposto). Tais informações poderão ser obtidas ou através de simulações (etapa 5B) ou através da execução do código fonte na plataforma alvo. Por fim, novos testes e verificações nos modelos poderão ser realizadas. Um exemplo de tais testes são as análises de escalonabilidade.

Por ser um aspecto fundamental para garantir a qualidade do sistema a ser desenvolvido e por esta razão, ser utilizada em várias etapas do processo de desenvolvimento, a verificação será objeto de uma atenção especial na seção seguinte.

6.4 Verificação Formal de Modelos

A verificação formal é uma técnica que permite confrontar e comparar a descrição operacional (ou de comportamento) de um sistema com as propriedades esperadas para este, usando métodos formais. As abordagens mais comumente utilizadas são: a verificação dedutiva por prova de teoremas, a verificação de modelos (*model checking*) que busca a satisfação de propriedades no modelo de comportamento do sistema e a verificação por equivalência entre modelos, a qual analisa a equivalência entre o modelo do comportamento e o modelo das propriedades.

O emprego da verificação formal de propriedades é praticamente obrigatório em várias etapas do processo de desenvolvimento dos SCAs, em particular os críticos, pois somente assim é possível dar alguma garantia de segurança (*safety*) aos resultados do

projeto. Atualmente existe uma série de ambientes e ferramentas que permitem o emprego de tais técnicas, nos mais diferentes níveis de abstração. Neste contexto, é possível destacar entre outras, o ambiente SCADE/Lustre [2] (ESTEREL-TECHNOLOGIES, 2011) e as ferramentas de verificação TINA/SELT [3] (BERTHOMIEU; RIBET; VERNADAT, 2004) e UPPAAL [4] (BEHRMANN; DAVID; LARSEN, 2004).

O ambiente SCADE/Lustre, muito utilizado na área aeroespacial e que se baseia na linguagem síncrona Lustre (HALBWACHS, 1993), é uma ferramenta comercial capaz de trabalhar num alto nível de abstração. Em consequência, SCADE tem potencial para ser utilizado em várias das etapas do processo de desenvolvimento apresentado anteriormente, em particular nas primeiras deste. Ferramentas de verificação como TINA/SELT e UPPAAL são ferramentas que trabalham num menor grau de abstração e tem como entrada linguagens formais (respectivamente Rede de Petri Temporizada (BERTHOMIEU; DIAZ, 1991) e Autômatos Temporizados (ALUR, 1999) com as quais os engenheiros de processo e de controle estão em geral pouco familiarizados. Por esta razão, o uso isolado destas ferramentas parece limitado para o desenvolvimento de sistemas complexos e nos quais várias competências (engenheiros de processos e de controle, engenheiros de software entre outros) são necessárias.

A seguir o ambiente TOPCASED [5] (FARAIL et al., 2006) é apresentado. É dado maior destaque às características que suportam a verificação de propriedades, a qual se baseia na técnica de verificação de modelos (CLARK; YUILLE, 1990).

6.4.1 Cadeia de verificação do ambiente TOPCASED

Devido à diversidade de linguagens de projeto existentes (UML, SysML, AADL, SDL, etc.) e de formalismos das ferramentas de verificação (rede de Petri, autômato, álgebra de processo, etc.), o ambiente TOPCASED definiu um passo intermediário de transformação de modelos.

Para tanto, é utilizada a linguagem FIACRE (BERTHOMIEU et al., 2008). A cadeia de verificação tem consequentemente dois níveis de transformação: um nível da linguagem de modelagem de domínio específico DSL para a linguagem FIACRE e, outro nível, de FIACRE para a linguagem de verificação (por exemplo sistema de transição temporizado TTS, utilizado pela ferramenta TINA), como mostrado na Figura 6.9.

[2] http://www.esterel-technologies.com/products/scade-suite/
[3] http://www.laas.fr/tina
[4] http://www.uppaal.org
[5] http://www.topcased.org

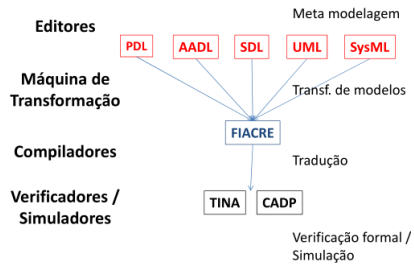


Figura 6.9: Cadeia de Verificação do TOPCASED

O principal benefício da utilização da linguagem FIACRE é a redução da lacuna semântica entre as linguagens de alto nível e os formalismos para ferramentas de verificação. A definição de uma única semântica para diferentes cadeias de verificação torna mais fácil a introdução de novas linguagens de alto nível e de novas ferramentas de verificação na cadeia como um todo. A linguagem FIACRE facilita a utilização da ferramenta de verificação TINA do ambiente TOPCASED em várias aplicações: SCAs críticos programados na linguagem AADL, CLPs programados com as linguagens da norma IEC 61131-3 (entre outras, o diagrama *ladder*), e também sistemas hipermídia interativos escritos na linguagem de domínio *Nexted Context Language* (NCL) (SOARES; BARBOSA, 2011). A seguir são detalhadas as principais características da linguagem FIACRE, a ferramenta de verificação TINA, uma ferramenta assistente para especificar propriedades e uma ferramenta de visualização dos resultados da verificação.

A linguagem FIACRE

FIACRE é uma linguagem fortemente tipada que oferece uma representação formal de aspectos comportamentais e temporais de um sistema. Os construtores sintáticos básicos de FIACRE são process e component. Um process é definido por um conjunto de estados e de transições. Para cada estado, uma expressão especifica as transições que determinam a passagem ao estado seguinte. Estas expressões são escritas por meio de construções determinísticas similares àquelas encontradas nas linguagens de programação clássicas (atribuição, condição, composição sequencial, laços, etc), construções não determinísticas (escolha, atribuição não determinística) e eventos de comunição sobre portas. Já um *component* é definido como a resultante de uma composição paralela de componentes e/ou processos, que se comunicam através de portas síncronas e de variáveis compartilhadas. A sintaxe dos componentes permite restringir o modo

de acesso e a visibilidade das variáveis compartilhadas e das portas. A associação de restrições temporais às comunicações sobre as portas e a definição de prioridades entre eventos de comunicação são características que podem também ser representadas nos componentes.

A ferramenta de verificação TINA

A ferramenta TINA (BERTHOMIEU; RIBET; VERNADAT, 2004) foi utilizada para editar e analisar Redes de Petri, Redes de Petri Temporizadas e Sistemas de Transição Temporizados (TTS). Além das características para editar e analisar estes modelos (presentes também em outras ferramentas), TINA tem como componentes uma ferramenta para abstração do espaço de estados e uma ferramenta de verificação de modelos denominada SELT.

A abstração do espaço de estados permite preservar classes de propriedades específicas tais como ausência de bloqueios (*deadlock*), propriedades temporais lineares ou bissimilaridades. Para sistemas não-temporizados, espaços de estados abstratos ajudam a evitar a explosão combinacional dos estados do sistema. Para sistemas temporizados, TINA oferece diversas abstrações baseadas em classes de estados, preservando propriedades de alcançabilidade, de linearidade e de ramificação.

Abstrações de espaço de estados são oferecidas em diversos formatos, permitindo o uso de TINA com diferentes ferramentas de verificação de modelos. A ferramenta SELT, que faz parte de TINA, além de ser capaz de verificar propriedades mais gerais (limitação de recursos, ausência de bloqueio, vivacidade (*liveness*)), também implementa um verificador de lógica temporal conhecido como State/Event LTL (SAGAR et al., 2004), o qual suporta tanto propriedades de estados como de transições.

O arcabouço para a verificação de modelos é uma estrutura de Kripke (CLARK; YUILLE, 1990) (o grafo de classe de estados construído pela ferramenta TINA). As fórmulas State/Event-LTL permitem expressar uma ampla variedade de propriedades referentes a estados e/ou transições. As fórmulas são avaliadas nos caminhos de execução da estrutura de Kripke, sendo construídas a partir de operadores do tipo X (no próximo passo), G (sempre) e F (no futuro). Seguem alguns exemplos de fórmulas:

X p: p é verdadeiro no próximo passo (*next*);

G p: p é verdadeiro sempre (*globally*);

F p: p é verdadeiro no futuro (*eventually*);

p U q: p é verdadeiro até que q o seja, q acontecendo no futuro (*until*);

p W q: p é verdadeiro até que q o seja, q podendo não acontecer no futuro (*weak until*).

Na lógica Temporal LTL, o tempo tem uma representação implícita. No caso da verificação de propriedades que necessitam incluir o tempo de forma explicita, podem ser utilizadas lógicas temporais temporizadas como TPTL (ALUR; HENZINGER, 1994) ou observadores temporais que observem o comportamento temporal do sistema e são associados a formulas LTL de alcançabilidade para poder verificar a correção deste (HALBWACHS; LAGNIER; RAYMOND, 1994). O uso destes observadores, geralmente escritos na mesma linguagem que o sistema a verificar e não interferindo nele, é o mais simples e usual pois utiliza os mesmos verificadores que na Lógica Temporal, em nosso caso LTL. Esta técnica vem sendo utilizada para uma ampla classe de propriedades de tempo-real a serem verificadas.

Ferramenta assistente para especificar propriedades

Com o objetivo de facilitar a tarefa do usuário para expressar as propriedades esperadas, foi desenvolvida uma ferramenta para auxiliar a especificação de propriedades em alto nível (no nível da linguagem de usuário) (OLIVEIRA; FARINES; BECKER, 2011). Para tanto, realizou-se uma classificação das propriedades, inicialmente baseando-se nos padrões definidos em (DWYER; AVRUNIN; CORBETT, 1999). Esta classificação foi modificada posteriormente por uma análise mais ampla, excluindo algumas propriedades pouco utilizadas e inserindo a propriedade de justiça *fairness*. As propriedades foram então organizadas segundo seis tipos de padrões:

- Ausência: expressa o desejo de garantia da segurança de um sistema (*safety*), evitando, por exemplo, um estado indesejável;
- Existência: afirma que um estado certamente é atingido, e inclui a propriedade de alcançabilidade (*reachability*);
- Justiça (*fairness*): expressa que algo ocorre infinitamente muitas vezes (*infinitely often*); a vivacidade (*liveness*) pode ainda ser testada a partir desta propriedade;
- Universalidade: permite representar invariantes durante a análise;
- Resposta: expressa que determinado estado sempre implica a ocorrência de algum outro estado;
- Precedência: expressa que determinado estado somente pode ser atingido depois da ocorrência de um outro estado.

Para cada um destes tipos de padrão foram definidos cinco escopos de análise, que restringem em que parte da execução do modelo um dado padrão será analisado.

Os escopos de análise são: *global, antes, depois, entre* e *enquanto não*. Além destes escopos temporais, são incluídas duas opções que permitem especificar as propriedades de ausência total de bloqueio e de alcançabilidade.

A ferramenta apresenta os padrões ao usuário sob a forma de uma estrutura em árvore e pode ser diretamente adaptada para cada DSL. Conforme a propriedade selecionada, é apresentada uma janela personalizada, contendo o significado da propriedade em linguagem natural e caixas de texto para inserir os estados correspondentes ao escopo de análise desejado, de forma a completar a especificação da propriedade. A primeira versão desta ferramenta foi desenvolvida para as propriedades de sistemas escritos na linguagem AADL. A Figura 6.10 apresenta a interface desta versão da ferramenta assistente. O exemplo utilizado nesta figura para ilustrar esta ferramenta e também apresentado na figura seguinte é o exemplo clássico do Marcador de Passos ("PaceMaker") conforme definido em (OLIVEIRA; FARINES; BECKER, 2011).

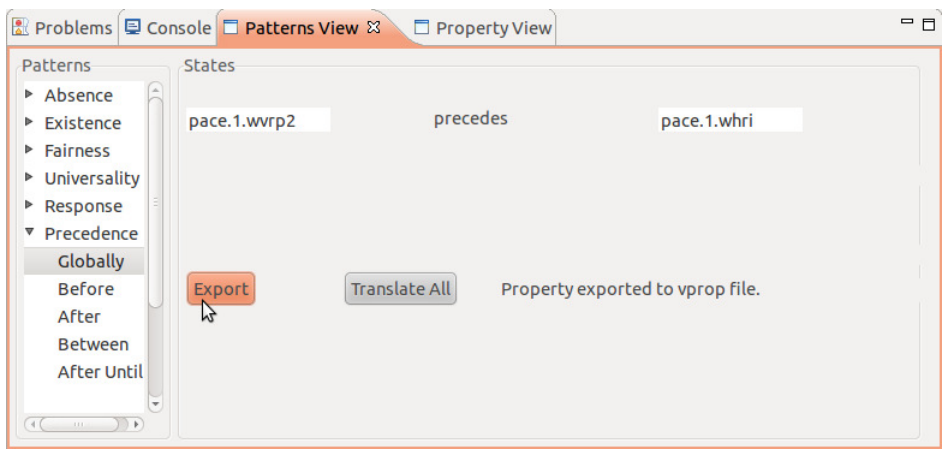


Figura 6.10: Interface da ferramenta assistente de propriedades

Ferramenta de visualização dos resultados da verificação

A visualização dos resultados da verificação pelo projetista na linguagem que ele utiliza usualmente é um aspecto indispensável para garantir o sucesso da abordagem de verificação proposta. Os resultados da ferramenta de verificação de modelos SELT, em particular os contraexemplos da não-verificação de uma propriedade, são geralmente apresentados na linguagem de entrada da ferramenta. No caso de TINA/SELT, o contraexemplo é apresentado em sistemas de transição temporizados TTS, de difícil entendimento pelo projetista. Para que o contraexemplo possa ser útil para o diagnóstico e

a correção posterior do erro detetado, torna-se necessário transformar o contraexemplo obtido em TTS na linguagem de uso do projetista ou num formato facilmente compreensível por ele, como por exemplo diagramas de tempo.

A ferramenta (OLIVEIRA; FARINES; BECKER, 2011) permite retornar os resultados da verificação na linguagem do projetista e visualizá-los nesta bem como a simulação de contra-exemplos no nível da linguagem do projetista. Construiu-se uma primeira versão desta ferramenta para a linguagem AADL, sendo que atualmente esforços similares estão sendo feitos para as linguagens do padrão IEC 61131-3 para CLP e para uma das linguagens padronizadas pela ITU para sistemas hipermídia, NCL.

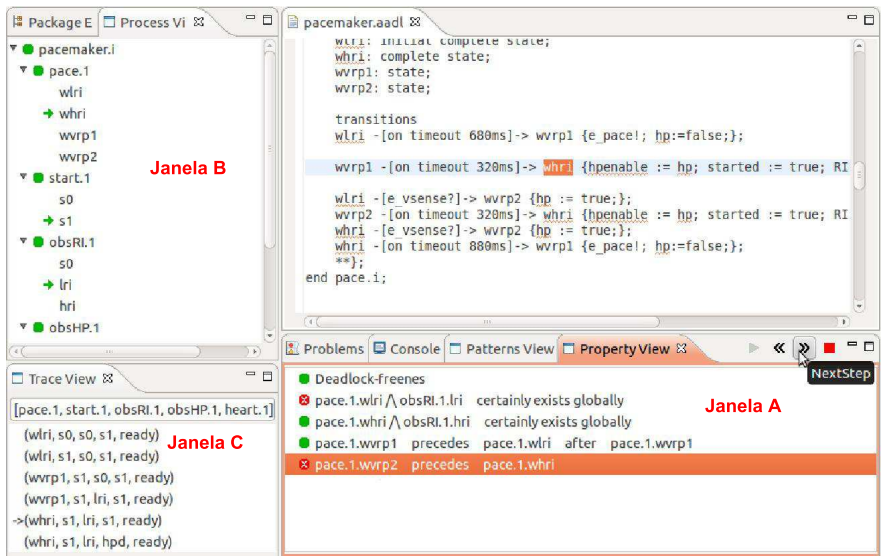


Figura 6.11: Interface de visualização de resultados

A Figura 6.11 ilustra a ferramenta proposta para AADL. Ela apresenta uma interface com janelas interelacionadas que permitem visualizar os seguintes pontos:

- o resultado de cada propriedade de verificação, apresentada em linguagem natural, conforme especificado na ferramenta de especificação de propriedades (janela A);
- os componentes do modelo, com os seus respectivos estados, na linguagem do usuário (janela B);
- o contra-exemplo para cada propriedade verificada como falsa, permitindo a sua execução (janela C).

Para uma propriedade verificada como falsa, a simulação passo-a-passo do respectivo contra-exemplo é apresentada na janela C (TraceView), sendo que na janela B (Package) o estado ativo de cada componente do modelo pode ser visualizado numa linguagem compreensível para o projetista .

6.4.2 Um exemplo ilustrativo de verificação de código AADL

Atualmente é possível realizar a verificação de três tipos de propriedades em modelos AADL: (i) propriedades implícitas analisadas pelo tradutor e que levam a bloqueios quando não satisfeitas; (ii) propriedades de usuários, especificadas através de observadores em AADL; (iii) propriedades especificadas diretamente em lógica temporal.

No que diz respeito às propriedades implícitas, no momento duas delas são analisadas pelo tradutor utilizado:

- **Escalonabilidade**: esta análise considera *threads* com prioridades fixas, definidas em tempo de projeto. O tradutor gera eventos sinalizando prazos (*deadlines*), sendo que um erro é sinalizado caso o evento ocorra enquanto a respectiva *threads* estiver ativa.
- **Buffer overflows**: a AADL define a propriedade *Overflow_Handling_Protocol*, a qual especifica o que fazer em caso de overflow. Caso a capacidade do *buffer* seja excedida, algum dado deverá ser descartado.

Para verificar propriedades temporais, como por exemplo um tempo limite de resposta, é possível utilizar *threads* AADL atuando como observadores de propriedades de tempo real. O componente a ser verificado é ligado a um observador, o qual exerce o papel de supervisor das duas respostas. Para exemplificar, criou-se no contexto do SEAV, um observador chamado `EnvironmentThread` para ser ligado ao componente `ParkingManeuver`. Este observador verifica se o sinal `highSpeed` é emitido, 10 ms após a velocidade se tornar maior que zero. Caso contrário, o estado de erro `err` é alcançado. Ele também verifica se o sinal `abort` é enviado caso o motorista movimente a direção do veículo. Através da ferramenta SELT mostrou-se que o estado `err` é inalcançável.

```
thread implementation EnvironmentThread.imp
annex behavior_specification {**
states
 s0: initial complete state;
 s1,s2,err: complete state;
transitions
```

```
 s0 - [ ] -> s0 { speed!(0); rangle!(0); };
 s0 - [finished?] -> s0;
 s0 - [ ] -> s1 { speed!(10); };
 s0 - [ ] -> s2 { wheelMoved!; };
 s1 - [highSpeed?] -> s0;
    -- detected in less than the 10ms period
 s1 - [on highSpeed'count = 0] -> err;
 s2 - [abort?] -> s0;
 s2 - [on abort'count=0] -> err;
**};
end EnvironmentThread.imp;
```

Deve-se realçar que tempos de resposta podem ser inseridos diretamente no modelo AADL como propriedade da especificação de fluxo, sendo verificada implicitamente. Entretanto, isso se torna difícil caso o tempo de resposta seja maior que o período mínimo do sinal de entrada. Aqui o observador pressupõe que o tempo de resposta não muda enquanto aguarda pelo sinal `highSpeed`.

A terceira forma de se fazer verificação na abordagem proposta é através da utilização de formulas da lógica temporal LTL (escritas diretamente pelo usuário ou usando a ferramenta assistente apresentada anteriormente). Tais propriedades são enviadas à ferramenta SELT que, com base no modelo do sistema, faz as devidas verificações. No caso do SEAV, foram definidas as seguintes propriedades:

- Se a velocidade for muito alta (FR2.2), a interface não consegue receber a mensagem de "vaga encontrada" (`found`) enquanto a busca não for reinicializada.

  $$G\,(\texttt{highSpeed} \Rightarrow (\neg\texttt{found}\,\mathbf{W}\,\texttt{startSearch}))$$

  Na verdade essa propriedade não é satisfeita dado que para considerar a informação de velocidade e abortar o processo é necessário um ciclo. Utilizou-se o evento de hiperperíodo *H* para reformular a propriedade da seguinte maneira: se a velocidade for muito alta, iniciando no próximo sinal do hiperperíodo, não se consegue receber a mensagem `found` a não ser que `startSearch` seja acionado.

  $$G\,\texttt{highSpeed} \Rightarrow (\neg\texttt{H}\,\mathbf{U}\,\texttt{H} \wedge (\neg\texttt{found}\,\mathbf{W}\,\texttt{startSearch}))$$

- É possível estacionar o veículo, i.e., existe um caminho de execução que leva ao estado onde o veículo é estacionado. A expressão é feita através de uma proprie-

dade negada (observa-se que a negação é necessária na linguagem temporal LTL para poder verificar a alcançabilidade): não é verdade que em toda execução o sinal `finished` nunca é enviado.

$$\texttt{Parking} \not\models FG\,\neg\texttt{finished}$$

Esta mesma fórmula também indica que o veículo pode ser estacionado um número ilimitado de vezes.

Por fim, um detalhe importante de ser evidenciado é que existem propriedades que não são possíveis de serem expressas na lógica LTL nem na lógica CTL (CLARK; YUILLE, 1990)(outro tipo de lógica temporal) como, por exemplo, a constatação de que a interface com o usuário pode ser reinicializada pelo próprio usuário independentemente do estado do sistema. Uma maneira de se resolver esse problema é através da utilização do *Mu-calculus* (CLARK; YUILLE, 1990) para expressar propriedades que podem ser verificadas em modelos atemporais através da ferramenta `muse`, também presente no ambiente TINA.

Finalmente, é também possível codificar uma estratégia de tempo real vencedora utilizando o anexo comportamental AADL (SAE, 2006) e então verificar se o estado inicial é alcançável utilizando uma propriedade LTL baseada no autômato abstrato gerado. No exemplo do SEAV, isso seria trivial, considerando que um comando de usuário poderia ser sempre utilizado.

6.5 Considerações finais

Este capítulo apresentou conceitos e técnicas relacionados ao desenvolvimento de sistemas de controle e automação, pela abordagem de Engenharia Dirigida por Modelos (MDE), que é uma metodologia de desenvolvimento de sistemas que permite construir uma aplicação, ou parte dela, a partir de modelos computacionais.

A seção 6.1 discutiu sobre MDE e transformação de modelos e a seção 6.2 apresentou um processo para o desenvolvimento de transformações de modelos desenvolvida em (MAGALHAES, 2016).

A seção 6.3 apresentou uma proposta de processo de desenvolvimento MDE para sistemas de controle e automação, em particular quando estes apresentam características de criticidade. Esta proposta é ilustrada por um exemplo de um sistema de estacionamento automático de veículos. Finalmente, a seção 6.4 deste capítulo tratou

de discutir questões relacionadas com a verificação formal de propriedades, ilustradas pelo mesmo exemplo.

Referências Bibliográficas

ABENI, L.; BUTTAZZO, G. Resource reservation in dynamic real-time systems. Real-Time Systems, [Upper Saddle River], v. 27, n. 2, p. 123-167, 2004.

AFANASYEV, A. et al. Host-to-host congestion control for TCP. IEEE Communications Society, Washington, DC, v. 12, n. 3, p. 304-342, May 2010.

AGUIRRE, L. A. Introdução à identificação de sistemas: técnicas lineares e não lineares aplicadas aos sistemas reais. 3. ed. Belo Horizonte: Editora UFMG, 2007.

AKYILDIZ, I. F. et al. A survey on sensor networks. IEEE Communications Magazine, New York, Aug 2002.

AL-KARAKI, J. N.; KAMAL, A. E. Routing techniques in wireless sensor networks: a survey. IEEE Wireless Communications, New York, v. 11, n. 6, p. 6-28, p. 2004.

ALIZADEH, F.; GOLDFARB, D. Second-order cone programming. Mathematical Programming, [S.l.], v. 95, n. 1, p. 3-51, 2003.

ALUR, R. Timed automata. Theoretical Computer Science, [Amsterdam]: Springer, v. 126, p. 183-235, 1994.

ALUR, R.; HENZINGER, T. A. A Really Temporal Logic. Journal of ACM, [New York], v. 41, n. 1, p. 181-203, 1994.

ALVISI, L. et al. Fault detection for byzantine quorum systems. IEEE Transactions on Parallel and Distributed Systems, New York, v. 12, n. 9, p. 996-1007, Sept. 2001.

AMBÜHL, C. et al. The range assignment problem in non-homogeneous static ad-hoc networks. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 18., 2004, [Los Alamitos]. Proceedings... [Los Alamitos: IEEE Computer Society], 2004.

ANDERSSON, B. Uniprocessor EDF scheduling with mode change. In: BAKER, T. P.; BUI, A.; TIXEUIL, S. (Ed.). Principles of Distributed Systems. [Berlin]: Springer, 2008. p. 572-577.

ANTA, A.; TABUADA, P. To sample or not to sample: Self-triggered control for nonlinear systems. IEEE Transactions on Automatic Control, [S.l.], v. 55, p. 2030-2042, 2010.

ARAÚJO, G. M. de; BECKER, L. B. A network conditions aware geographical forwarding protocol for real-time applications in mobile wireless sensor networks. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 25., 2011, Biopolis. Proceedings... [S.l.]: IEEE Computer Society, 2011. p. 38-45.

ARAÚJO, G. M. de. Uma contribuição para protocolos de redes de sensores com provimento de qualidade de serviço em ambientes móveis. [S.l.], 2011.

ARAÚJO, J. et al. Self-triggered control for wireless sensor and actuator networks. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING IN SENSOR SYSTEMS, 7., 2011. Proceedings... [S.l.: s.n.], 2011. p. 1-9.

ASTRÖM, K. J.; WITTENMARK, B. Adaptive Control. 2nd. Mineol: Dover Publications, 1995.

ASZTALOS, M.; LENGYEL, L.; LEVENDOVSZKY, T. Towards automated, formal verification of model transformation. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, 3., 2010. Proceedings... [S.l.: s.n.], 2010. p. 15-24.

ATL. ATL Model Transformation Language. Disponivel em:<http://www.eclipse.org/atl/>. Acesso em: 31 Dec. 2011

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: a survey. Computer Networks, [S.L.], v. 54, n. 15, p. 2787-2805, 2010.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, New York, v. 1, n. 1, p. 11-33, Jan./Mar. 2004.

BAILLIEUL, J.; ANTSAKLIS, P. J. Control and communication challenges in networked real-time systems. Proceedings of the IEEE, New York, v. 95, n. 1, p. 9-28, Mar. 2007.

BARBOZA, F.; ASSIS, F. A localized algorithm based on minimum cost arborescences for the MECBS problem with asymmetric edge costs. In: SIMPLOT-RYL, D. et al. (Ed.). Ad Hoc Networks. New York: Springer, 2011. p. 223-238.

BECKER, L. B. et al. Development process for critical embedded systems. In: BRAZILIAN WORKSHOP ON EMBEDDED SYSTEMS, 1., 2010. Proceedings ... [S.l.: s.n.], 2010. p. 95-108.

BEHRMANN, G.; DAVID, R.; LARSEN, K. G. A tutorial on UPPAAL. In: BERNARDO, M.; CORRADINI, F. Formal Methods for the Design of Computer, Communication, and Software Systems. Berlim: Springer-Verlag, 2004. p. 200-237. (Lecture Notes in Computer Science, v. 3185).

BELDIMAN, O.; BUSHNELL, L.; WALSH, G. C. Predictors for networked control systems. American Control Conference. [S.l.], v. 4, p. 2347-2351, 2000.

BERNAT, G.; BURNS, A.; LLAMOSI, A. Weakly hard real-time systems. IEEE Transactions on Computers. IEEE Computer Society, Washington, v. 50, n. 4, p. 308-321, 2001.

BERTHOMIEU, B. et al. FIACRE: an intermediate language for model verification in the TOPCASED environment. In: 4th European Congress on Embedded Real-Time Software ERTS'08 (Toulouse, France). [S.l.: s.n.], 2008.

BERTHOMIEU, B.; DIAZ, M. Modeling and verification of time dependent systems using time petri nets. IEEE Transactions on Software Engineering, [New York], v. 17, n. 3, p. 259-273, Mar. 1991.

BERTHOMIEU, B.; RIBET, P.; VERNADAT, F. The tool TINA -construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research, [London], v. 42, n. 14, 2004.

BERTIER, M.; MARIN, O.; SENS, P. Implementation and performance evaluation of an adaptable failure detector. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2002, Washington. Proceedings... Washington: IEEE Computer Society, 2002. p. 354-363.

BERTIER, M.; MARIN, O.; SENS, P. Performance analysis of a hierarchical failure detector. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2003., San Francisco. Proceedings... San Francisco: IEEE Computer Society, 2003. p. 635-644.

BERTSEKAS, D. P. Nonlinear Programming. Belmont: Athena Scientific, 1995.

BEZIVIN, J. Model transformations? transformation models? In: INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, 9., 2006, Genova. Proceedings...Berlin: Springer-Verlag, 2006. p. 440-453.

BIRGE, J. R.; LOUVEAUX, F. Introduction to Stochastic Programming. [New York]: Springer-Verlag, 1997.

BLOUGH, D. M. et al. Topology control with better radio models: implications for energy and multi-hop interference. In: INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS AND SIMULATION OF WIRELESS AND MOBILE SYSTEMS, 2005, Montreal. Proceedings... New York: ACM, 2005. p. 260-268.

BOYD, S.; VANDENBERGHE, L. Convex Optimization. Cambridge: Cambridge University Press, 2004.

BRAGA, H.; ASSIS, F. A topology control algorithm for interference and energy efficiency in wireless sensor networks. In: In: FREY, H.; LI, X., RUEHRUP, S. (Ed.). Ad-hoc, Mobile, and Wireless Networks. Berlin: Springer, 2011. v. 6811, p 86-99.

BRANICKY, M. S.; PHILLIPS, S. M.; ZHANG, W. Stability of networked control systems: Explicit analysis of delay. In: AMERICAN CONTROL CONFERENCE, 2000, Chicago. Proceedings... Evanston: American Automatic Control Council, 2000. v. 4, p. 2352-2357.

BURKHART, M. Analysis of interference in ad hoc networks. 2003. 48 f. Thesis – Swiss Federal Institute of Technology Zurich, Zürich, 2003.

BURKHART, M. et al. Does topology control reduce interference? In: MOBIHOC. New York, NY, USA: [s.n.], 2004. p. 9-19.

BUTTAZZO, G.; ABENI, L. Adaptive workload management through elastic scheduling. Real-Time Systems. Kluwer Academic Publishers, [S.l.], v. 23, n. 1/2, p. 7-24, 2002.

CACCAMO, M.; BUTTAZZO, G.; SHA, L. Capacity sharing for overrun control. In: REAL-TIME SYSTEMS SYMPOSIUM, 21., 2002, Orlando, Proceedings... Los Alamitos: IEEE Computer Society, 2000. p. 295-304.

CACCAMO, M.; BUTTAZZO, G. C.; THOMAS, D. C. Efficient reclaiming in reservation-based real-time systems with variable execution times. IEEE Transactions on Computers, [S.l.], v. 54, n. 2, p. 198-213, 2005.

CAGALJ, M.; HUBAUX, J.-P.; ENZ, C. Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues. In: MOBICOM'02. New York: ACM, 2002. p. 172-182.

CALAMONERI, T. et al. Minimum-energy broadcast in random-grid ad-hoc networks: Approximation and distributed algorithms. In: INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS AND SIMULATION OF WIRELESS AND MOBILE SYSTEMS, 11., 2008, Vancouver. Proceedings.... New York: ACM Press, 2008. p. 354-361.

CALINESCU, G. et al. Network lifetime and power assignment in ad hoc wireless networks. In: DI BATTISTA, G.; ZWICK, U. (Ed.). Algorithms - ESA 2003. [S.l.]: Springer Verlag, 2003. (LNCS, v. 2832).

CAMBRUZZI, E. et al. A cluster management system for vanets. International Journal of Intelligent Transportation Systems Research, [New York], v. 14, n. 2, p. 115-126, 2016.

CAMP, J.; KNIGHTLY, E. W. The IEEE 802.11 s extended service set mesh networking standard. IEEE Communications Magazine, New York, v. 46, n. 8, p. 120-126, 2008.

CAO, J.; ZHONG, S.; HU, Y. Novel delay-dependent stability conditions for a class of mimo networked control systems with nonlinear perturbation. Applied Mathematics and Computation, New York, v. 197, n. 2, p. 797-809, 2008.

CARTIGNY, J. et al. Localized LMST and RNG based minimum-energy broadcast protocols in ad hoc networks. Ad Hoc Networks, [Amsterdam], v. 3, n. 1, 2005.

CARTIGNY, J.; SIMPLOT-RYL, D.; STOJMENOVIC, I. Localized minimum-energy broadcasting in ad-hoc networks. In: Procs. of INFOCOM. [S.l.: s.n.], 2003.

CENA, G.; VALENZANO, A.; VITTURI, S. Hybrid wired/wireless networks for real-time communications. IEEE Industrial Electronics Magazine, v. 2, n. 1, p. 8-20, 2008.

CERVIN, A. et al. The jitter margin and its application in the design of real-time control systems. In: INTERNATIONAL CONFERENCE ON REAL-TIME AND EMBEDDED COMPUTING SYSTEMS AND APPLICATIONS, 10., 2004, Washington. Proceedings... Los Alamitos: IEEE Computer Society, 2004.

CERVIN, A.; OHLIN, M.; HENRIKSSON, D. Simulation of networked control systems using truetime. In: INTERNATIONAL WORKSHOP ON NETWORKED CONTROL SYSTEMS: TOLERANT TO FAULTS. 3., 2007, Nancy. Proceedings... [S.l.: s.n.], 2007.

CHANDRA, T. D.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, New York, v. 43, n. 2, p. 225-267, Mar. 1996.

CHARNES, A.; COOPER, W. W. Deterministic equivalents for optimizing and satisficing under chance constraints. Operations Research, Hanover, v. 11, n. 1, p. 18-39, 1963.

CHEN, F.; GERMAN, R.; DRESSLER, F. Towards IEEE 802.15.4e: A study of performance aspects. In: CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS, 8., 2010, Mannheim. Proceedings... Piscataway, N.J.: IEEE] 2010. p. 68-73.

CHEN, W.; TOUEG, S.; AGUILERA, M. K. On the quality of service of failure detectors. IEEE Transactions on Computers. IEEE Computer Society, Los Alamitos, v. 51, n. 2, p. 561-580, May 2002.

CHEN, Y.; ZHANG, J.; MARSIC, I. Link-layer-and-above diversity in multi-hop wireless networks. IEEE Communications Magazine, New York, v. 47, n. 2, p. 118-124, 2009.

CHENG, A. M. K. Real-Time systems: scheduling, analysis, and verification. New Jersey: John Wiley & Sons, 2002.

CHENG, X. et al. Strong minimum energy topology in wireless sensor networks: NP-completeness and heuristics. IEEE Transactions on Mobile Computing, New York, v. 2, n. 3, 248-256, 2003.

CHIPARA, O. et al. Real-time power-aware routing in sensor networks. In: IEEE. Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on. [S.l.], 2006. p. 83-92.

CHRISTIANO, B.; MENEZES, R.; COMICIO, T. On the specification, verification and implementation of model transformation with transformation contracts. In: 14th Brazilian Symposium, SBMF 2011. [S.l.]: Springer-Verlag, 2011.

CLARK, J.; YUILLE, A. Data fusion for sensory information processing systems. [S.l.]: Springer, 1990.

CLEMENTI, A. et al. On the complexity of computing minimum energy consumption broadcast subgraphs. In: FERREIRA, A.; REICHEL, H. (Ed.). STACS 2001. Berlin: Springer Verlag, 2001. (LNCS, v. 2010).

CLEMENTI, A. et al. A Worst-Case Analysis of an MST-based Heuristic to Construct Energy-Efficient Broadcast Trees in Wireless Networks. [S.l.], 2001.

CLEMENTI, A. E. F.; PENNA, P.; SILVESTRI, R. Hardness results for the power range assignment problem in packet radio networks. In: HOCHBAUM, D. et al. (Ed.): Randomization, Approximation, and Combinatorial Algorithms and Techniques, Third International Workshop on Randomization and Approximation Techniques in Computer Science, and Second International Workshop on Approximation Algorithms for Combinatorial Optimization Problems RANDOM-APPROX'99.1999. Berlin: Springer, 1999. p. 197-208. (Lecture notes in computer science, v. 1671).

COMMITTEE, I. C. S. Part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area network (LR-WPAN) - IEEE Std 802.15.4. 2006.

COMMITTEE, I. C. S. IEEE STD 802.15.4e-D0.01, Part 15.4: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), Amendment 5: Amendment to the MAC sub-layer. 2010.

COOPER, R. Introduction to Queuing Theory. 2nd. [New York]: North Holland, 1981.

CORMEN, T. H. et al. Introduction to Algorithms. 2nd. London: The MIT Press, 2001.

CRISTIAN, F. Understanding fault-tolerant distributed systems. Communications of the ACM, New York, v. 34, n. 2, p. 56-78, Feb. 1991.

CRISTIAN, F.; FETZER, C. The timed asynchronous distributed system model. IEEE Transactions on Parallel Distributed Systems, New York, v. 10, p. 642-657, June 1999.

CSERTAN, G. e t al. Viatra: Visual automated transformations for formal verification and validation of UML models. In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 17.,2002, Edinburgh. Proceedings... Los Alamitos: IEEE Computer Society, 2002.

DAI, S. H.; WANG, M. Reliability analysis in engineering applications. New York: Van Nostrand Reinhold, 1992.

DAMIAN, M.; JAVALI, N. Distributed construction of bounded-degree low-interference spanners of low weight. In: MOBIHOC. New York: [s.n.], 2008. p. 101-110.

DECOTIGNIE, J. D. Ethernet-based real-time and industrial communications. Proceedings of the IEEE, New York, v. 93, n. 6, p. 1102-1117, 2005.

DÉFAGO, X.; SCHIPER, A.; URBÁN, P. Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Computing Surveys, New York, v. 36, p. 372-421, December 2004.

DESWARTE, Y.; KANOUN, K.; LAPRIE, J. Diversity against accidental and deliberate faults. In: CONFERENCE ON COMPUTER SECURITY, DEPENDABILITY, AND ASSURANCE: FROM NEEDS TO SOLUTIONS. 1998. Proceedings... Washington, DC, USA: IEEE Computer Society, 1998. p. 171-181.

DIXIT, M. et al. Adaptare-FD: A dependability-oriented adaptive failure detector. In: SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS. NEW DELHI, INDIA: IEEE COMPUTER SOCIETY, 29., 2010, New Delhi. Proceedings... Los Alamitos: IEEE Computer Society, p. 141-147.

DOLEV, D. et al. Failure detectors in omission failure environments. In: ANNUAL ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 16., 1997, Santa Barbara. Proceeding.... New York: Association for Computing Machinery, 1997.

DOUDOU, A.; GARBINATO, B.; GUERRAOUI, R. Encapsulating failure detection: From crash to byzantine failures. In: BLIEBERGER, J., STROHMEIER, A. (Ed.). Reliable Software Technologies – Ada-Europe 2002. Ada-Europe. Berlin: Springer. p. 24-50. (Lecture Notes in Computer Science, v. 236).

DOUDOU, A. et al. Muteness failure detectors: Specification and implementation. In: EUROPEAN DEPENDABLE COMPUTING CONFERENCE. 3., Prague: Sprnger, 1999. Proceedings... (LNCS 1667), p. 71-87.

DWORK, C.; LYNCH, N.; STOCKMEYER, L. Consensus in the presence of partial synchrony. Journal of ACM, New York, v. 35, p. 288-323, Apr. 1988.

DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C. Patterns in property specifications for finite-state verification. In: 1st International Conference on Software Engineering ICSE'99. [S.l.: s.n.], 1999.

ECLIPSE. MOF Script. [S.l.], 2005. Disponível em: <http://www.eclipse.org/gmt/mofscript/> Acesso em: 1 Oct. 2016.

ELIA, N. Remote stabilization over fading channels. Systems & Control Letters, Amsterdam, v. 54, n. 3, p. 237-249, 2005.

ESTEREL-TECHNOLOGIES. SCADE Suite. [S.l.], 2011. <http://www.estereltechnologies.com/products/scade-suite/>. Acesso em: 1 Mar. 2017.

FALAI, L.; BONDAVALLI, A. Experimental evaluation of the QoS failure detectors on wide area network. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2005, Yokohama. Proceedings... Washington: IEEE Computer Society IEEE Computer Society, 2005. p. 624-633.

FAN, P. Improving broadcasting performance by clustering with stability for inter-vehicle communication. Vehicular Technology Conference, [S.l.], 2007.

FARAIL, P. et al. The TOPCASED project: Toolkit in OPen-source for Critical Applications and SystEms development. In: EUROPEAN CONGRESS ON EMBEDDED REAL TIME SOFTWARE – ERTS, 3., 2006, Toulouse. Proceedings... [S.l.: s.n.], 2006.

FELBER, P. The corba object group service: A service approach to object groups in CORBA. 1998. 179 f. Thèse (Doctorat) – Département D'Informatique, école Polytechnique Fédérale De Lausanne, Lausanne, 1998. Disponível em: <http://biblion.ep.ch/EPFL/theses/1998/1867/EPFL TH1867.pdf>. Acesso em: 4 Oct. 2011.

FELEMBAN, E.; EKICI, E. MMSPEED: multipath Multi-SPEED protocol for QoS guarantee of reliability and. Timeliness in wireless sensor networks, IEEE Transactions on Mobile Computing, New York, v. 5, n. 6, p. 738-754, 2006.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. Journal of ACM, New York, v. 32, p. 374-382, Apr. 1985.

FRANCA, R. B. et al. The AADL behaviour annex - experiments and roadmap. In: INTERNATIONAL CONFERENCE ON ENGINEERING COMPLEX COMPUTER SYSTEMS. 12., 2007, Auckland. Proceedings...Washington: IEEE Computer Society, 2007. p. 377-382.

FRANKLIN, G. F.; POWELL, J. D.; EMAMI-NAEINI. Feedback control of dynamic systems. 4th. [S.l.]: Prentice Hall, 2002.

FRIEDMAN, R.; TCHARNY, G.; LTD, I. Evaluating failure detection in mobile ad-hoc networks. Journal of Wireless and Mobile Computing, [S.l.], v. 1, n. 8, 2005.

FUSSEN, M.; WATTENHOFER, R.; ZOLLINGER, A. Interference arises at the receiver. In: INTERNATIONAL CONFERENCE ON WIRELESS NETWORKS, COMMUNICATIONS AND MOBILE COMPUTING, 2005, Proceedings... Piscataway: IEEE Operations Center, 2005. v. 1, p. 427-432.

GAO, H.; CHEN, T.; LAM, J. A new delay system approach to network-based control. Automatica, New York, v. 44, n. 1, p. 39-52, 2008.

GAO, Y.; HOU, J. C.; NGUYEN, H. Topology control for maintaining network connectivity and maximizing network capacity under the physical model. In: INFOCOM, 27., 2008, Phoenix. Proceedings...[S.l.: s.n] 2008. p. 1013-1021.

GAREY, M. R.; JOHNSON, D. S. Computers and Intractability: a guide to the theory of NP-Completeness. New York: W. H. Freeman & Co., 1979.

GÄRTNER, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Computing Surveys, New York, v. 31, p. 1-26, Mar. 1999.

GEORGES, J. P.; DIVOUX, T.; RONDEAU, E. Validation of the network calculus approach for the performance evaluation of switched ethernet based on industrial communications. In: IFAC WORLD CONGRESS, 16., 2005, Prague. Proceedings... Prague: International Federation of Automatic Control, 2005.

GEORGES, J.-P. et al. Use of upper bound delay estimate in stability analysis and robust control compensation in network control systems. IFAC Proceedings Volumes, [Oxford], v. 39, n. 3, p. 107-112, 2006.

GHOSH, S. K. Energy efficient broadcast in distributed ad-hoc wireless networks. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE AND ENGINEERING, 11., 2008, São Paulo. Proceedings... Los Alamitos: IEEE Computer Society, 2008. p. 394-401.

GORENDER, S.; MACÊDO, R. J. A.; RAYNAL, M. A hybrid and adaptative model for fault-tolerant distributed computing. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2005, Yokohama. Proceedings... Alamitos: IEEE Computer Press, 2005. v. 1. p. 412-421.

GORENDER, S.; MACÊDO, R. J. A. Consenso com recuperação no modelo partitioned synchronous. In: WORKSHOP DE TESTES E TOLERÂNCIA A FALHAS, 11., 2010, Gramado. Anais... Porto Alegre: Sociedade Brasileira de Computação, 2010. v. 1, p. 3-16.

GORENDER, S.; MACÊDO, R. J. A.; RAYNAL, M. An adaptive programming model for fault-tolerant distributed computing. IEEE Transactions on Dependable Secure Computing, New York, v. 4, p. 18-31, Jan. 2007.

GORENDER, S.; MACÊDO, R. J. A. Um modelo para tolerância a falhas em sistemas distribuídos com QoS. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 20., 2002, Búzios. Anais... Porto Alegre: Sociedade Brasileira de Computação, 2002. p. 277-292.

GUERRAOUI, R. et al. Consensus in asynchronous distributed systems: a concise guided tour. In: KRAKOWIAK, S.; SHRIVASTAVA, S. (Ed.). Advances in distributed systems: from algorithms to systems. Berlin: Springer, 2000. p. 33-47. (Lecture Notes in Computer Science, v. 1752).

GUNTER, Y.; WIEGEL, B.; GROSSMANN, H. Cluster-based medium access scheme for vanets. IEEE Intelligent Transportation Systems article. IEEE INTELLIGENT TRANSPORTATION SYSTEMS CONFERENCE, 2007, Seattle. Proceedings... Seattle, 2007. p. 343-348.

GUPTA, V. et al. On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage. Automatica, New York, v. 42, n. 2, p. 251-260, 2006.

GUPTA, V.; HASSIBI, B.; MURRAY, R. M. Optimal lqg control across packet-dropping links. Systems & Control Letters, Amsterdam v. 56, n. 6, p. 439-446, 2007.

HAENGGI, M. Opportunities and challenges in wireless sensor networks. In: ILYAS, M.; MAHGOUB, I. (Ed.). Handbook of Sensor Networks: compact wireless and wired sensing systems. Boca Raton: CRC Press, 2004.

HALBWACHS, N. Synchronous programming of reactive systems. Dordrecht: Kluwer Academic Publisher, 1993.

HALBWACHS, N.; LAGNIER, F.; RAYMOND, P. Synchronous observers and the verification of reactive systems. In: INTERNATIONAL CONFERENCE ON ALGEBRIC METHODOLOGY AND SOFTWARE TECHNOLOGY, 3., 1993, Enschede. Proceedings ... Enschede, 1993. p. 83-96.

HALEVI, Y.; RAY, A. Integrated communication and control systems: Parti - analysis. Journal of dynamic systems, measurement and control, New York, v. 110, n. 4, p. 367-373, 1988.

HAYKIN, S. Neural networks: a comprehensive foundation. 1st. New York: MacMillan, 1994.

HE, F.; GUO, K. Modeling and simulation of profibus-dp network control system. In: INTERNATIONAL CONFERENCE ON AUTOMATION AND LOGISTICS, 2008, Qingdao. Proceedings... Piscataway: IEEE Xplore, 2008. p. 1141-1146.

HE, T. et al. SPEED: A stateless protocol for real-time communication in sensor networks. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 23., 2003, Providence. Proceedings... Los Alamitos: IEEE Computer Society, 2003.

HEINZELMAN, J. K. W. R.; BALAKRISHNAN, H. Adaptive protocols for information dissemination in wireless sensor networks. In: INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, 5., 1999, Seattle. Proceedings... New York, N.Y. : ACM, 1999. p. 174-185.

HEINZELMAN, W. R.; CH, A.; BALAKRISHNAN, H. Energy-efficient communication protocol for wireless microsensor networks. In: INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 33., 2000, Maui. Proceedings... Los Alamitos : IEEE Computer Society Press, 2000. p. 3005-3014.

HELLERSTEIN, J. L. et al. Feedback control of computing systems. Canada: Wiley-Interscience, 2004.

HENRIKSSON, D.; CERVIN, A.; OHLIN, M. Truetime 1.5 - Reference manual. Sweden: Department of Automatic Control, Lund University, 2007. Disponível em: <http://www3.control.lth.se/truetime/>. Acesso em: 01 Mar. 2008.

HENRIKSSON, D.; CERVIN, A.; ÅRZÉN, K. E. Truetime: simulation of control loops under shared computer resources. In: IFAC WORLD CONGRESS ON AUTOMATIC CONTROL, 15., 2002, Barcelona. Proceedings.... Oxford: Published for the International Federation of Automatic Control by Pergamon, 2002.

HESPANHA, J. P.; NAGHSHTABRIZI, P.; XU, Y. A survey of recent results in networked control systems. Proceedings of the IEEE, New York, v. 95, n. 1, p. 138-162, 2007.

HONG, S. H. Scheduling algorithm of data sampling times in the integrated communication and control systems. IEEE Transactions on Control Systems Technology, New York, v. 3, n. 2, p. 225-230, 1995.

HORST, R.; PARDALOS, P. M.; THOAI, N. V. Introduction to global optimization. New York: Springer, 2000. (Nonconvex Optimization and Its Applications).

HORST, R.; TUY, H. Global optimization: deterministic approaches. Berlin: Springer, 2003.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing: degrees, models, and applications. ACM Computing Surveys, New York, v. 40, p. 1-28, Aug. 2008.

INDUSTRIAL WIRELESS COMMUNITY. Industrial wireless technology for the 21st century. [S.l.: s.n.], 2002.

INGELREST, F.; SIMPLOT-RYL, D. Localized broadcast incremental power protocol for wireless ad hoc networks. Rapports techniques, Rocquencourt, v. 14, 2008.

INGELREST, F.; SIMPLOT-RYL, D.; STOJMENOVIC, I. Optimal transmission radius for energy efficient broadcasting protocols in ad hoc and sensor networks. IEEE Transactions on Parallel and Distributed Systems, New York, v. 17, n. 6, p. 536-547, Jun. 2006.

INTANAGONWIWAT, C.; GOVINDAN, R.; ESTRIN, D. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In: MOBICOM. [S.l.]: ACM, 2000. p. 56-67.

IRWIN, G.; COLANDAIRAJ, J.; SCANLON, W. An overview of wireless networks in control and monitoring. In: HUANG, D.-S., LI, K.; IRWIN, G. W. (Ed.). Computational Intelligence. Berlin: Springer, 2006. p. 1061-1072.

ISA. ISA100.11a:2008 Draft standard Wireless systems for industrial automation: Process control and related applications. Alexander Drive, 2008.

ISHII, H. H-∞ control with limited communication and message losses. Systems & Control Letters, Amsterdam, v. 57, n. 4, p. 322-331, 2008.

JACOBSON, V. Congestion avoidance and control. ACM SIGCOMM Computer Communication Review, v. 18, n. 4, p. 314-329, Aug. 1988.

JAIN, R.; ROUTHIER, S. Packet trains: measurements and a new model for computer network traffic. IEEE Journal on Selected Areas in Communications, New York, v. 4, n. 6, p. 986-995, Sep. 1986.

JALOTE, P. Fault tolerance in distributed systems. New Jersey: Prentice Hall, 1994.

JIANG, X.; HAN, Q. L. Delay-dependent robust stability for uncertain linear systems with interval time-varying delay. Automatica, New York, v. 42, n. 6, p. 1059-1065, 2006.

JIN, Z. et al. A survey on position-based routing algorithms in wireless sensor networks. Algorithms, [S.l.], v. 2, n. 1, p. 158-182, fev. 2009.

JING, W.; LIQIAN, Z.; TONGWEN, C. An mpc approach to networked control design. In: Chinese Control Conference 2007. [S.l.: s.n.], 2007. p. 10-14.

JOHANSSON, T.; CARR-MOTYCKOVà, L. Reducing interference in ad hoc networks through topology control. In: DIALM-POMC. 2005,Cologne. Proceedings... New York: ACM, 2005. p. 17-23.

KALMAN, R. E. A new approach to linear filtering and prediction problems. Journal of basic Engineering, [S.l.], v. 82, n. 1, p. 35-45, 1960.

KAO, C. Y.; LINCOLN, B. Simple stability criteria for systems with time-varying delays. Automatica, New York, v. 40, n. 8, p. 1429-1434, 2004.

KENT, S. Model driven engineering. In: Integrated Formal Methods. Berlin: Springer Verlag, 2002. p. 286-298. (Lecture Notes in Computer Science, v. 2335)

KIHLSTROM, K. P.; MOSER, L. E.; MELLIAR-SMITH, P. M. Byzantine fault detectors for solving consensus. The Computer Journal, [S.l.], v. 46, n. 1, p. 16-35, Jan. 2003.

KIM, A. et al. When HART goes wireless: understanding and implementing the wireless hart standard. In: INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 13., 2008, Hamburg. Proceedings... Piscataway: IEEE Xplore, 2008.

KIM, K. D.; KUMAR, P. R. Cyber-physical systems: A perspective at the centennial. Proceedings of the IEEE, New York, v. 100, p. 1287-1308, May 2012. Edition Special.

KIM, S.; BURGER, D.; CARRINGTON, D. An MDA approach towards integrating formal and informal modeling languages. In: Lecture Notes in Computer Science Volume, [S.l.]: Springer Verlag, 2005. v. 3582, p. 448-464.

KOPETZ, H. Real-time systems: Design principles for distributed embedded applications. 1st. Norwell: Kluwer Academic Publishers, 1997.

KOREN, G.; SHASHA, D. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In: REAL-TIME SYSTEMS SYMPOSIUM, 16., 1995. Pisa. Proceedings... Los Alamitos: IEEE Computer Society Press, 1995. p. 110-117.

KULIK, J.; HEINZELMAN, W. Negotiation-based protocols for disseminating information in wireless sensor networks. Wireless Networks, [S.l.], v. 8, p. 169-185, 2002.

KURTEV, I. Adaptability of Model Transformations. 2005. Thesis – University of Twente, Enschede, 2005.

LAMPORT, L.; LYNCH, N. Chapter on distributed computing: methods and models. Cambridge, 1989. Disponível em: <http://research.microsoft.com/users/lamport/pubs/lamport-chapter.pdf>. Acesso em: 4 Oct. 2011.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The byzantine generals problem. ACM Transaction on Programming Languages Systems, New York, v. 4, n. 3, p. 382-401, July 1982.

LANDSIEDEL, O.; WEHRLE, K.; GOTZ, S. Accurate prediction of power consumption in sensor networks. In: WORKSHOP ON EMBEDDED NETWORKED SENSORS, 2., 2005, Sydney. Proceedings... Piscataway: IEEE Operations Center, 2005.

LANO K.; CLARK, D. Model transformation specification and verification. In: INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE. 8., 2008, Oxford. Proceedings... Piscataway: NJ IEEE, 2008.

LEE, K. C.; LEE, S. Performance evaluation of switched Ethernet for real-time industrial communications. Computer standards & interfaces, Amsterdam, v. 24, n. 5, p. 411-423, 2002.

LEMMON, M. Event-triggered feedback in control, estimation, and optimization. In: BEMPORAD, A.; Heemels, M.; VEJDEMO-JOHANSSON, M. Networked Control Systems. London: Springer London, 2010. p. 293-358. (Lecture Notes in Control and Information Sciences, v. 406).

LEON-GARCIA, A. Probability and Random Processes for Electrical Engineering. Reading, MA: Addison-Wesley Publishing Company, 1994.

LEROY, X. Java bytecode verification: Algorithms and formalizations. Journal of Automated Reasoning, Dordrecht, v. 30, n. 3, p. 235-269, 2003.

LI, X.-Y. et al. Interference-aware topology control for wireless sensor networks. In: ANNUAL IEEE COMMUNICATIONS SOCIETY CONFERENCE ON SENSOR AND ADHOC COMMUNICATIONS AND NETWORKS, 2., 2005, Santa Clara. Proceedings... Piscataway: IEEE 2005. p. 263-274.

LI, Y. et al. On the construction of a strongly connected broadcast arborescence with bounded transmission delay. IEEE Transactions on Mobile Computing, Piscataway, v. 5, n. 10, p. 1460-1470, 2006.

LIAN, F.-L.; MOYNE, J.; TILBURY, D. Network design consideration for distributed control systems. IEEE Transactions on Control Systems Technology, New York, v. 10, n. 2, p. 297-307, Mar. 2002.

LIAN, F. L.; MOYNE, J. R.; TILBURY, D. M. Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet. IEEE Control Systems Magazine, New York, v. 21, n. 1, p. 66-83, 2001.

LIANG, W. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In: INTERNATIONAL SYMPOSIUM ON MOBILE AD HOC NETWORKING & COMPUTING, 3., Lausanne. Proceedings... New York: ACM, 2002. p. 112 - 122.

LIMA, F.; MACÊDO, R. J. A. Adapting failure detectors to communication network load fluctuations using snmp and artificial neural nets. In: LATIN-AMERICAN CONFERENCE ON DEPENDABLE COMPUTING, 2., 2005, Salvador. Proceedings... Berlin: Springer-Verlag, 2005. p. 191-205.

LINDSEY, S.; RAGHAVENDRA, C. S. PEGASIS: Power-efficient gathering in sensor information systems. In: AEROSPACE CONFERENCE PROCEEDINGS, 2002, Montana. Proceedings... Piscataway: IEEE, 2002. v. 3.

LIPARI, G.; BARUAH, S. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 12., 2000, Stockholm, Proceedings... Los Alamitos: IEEE Computer Society, 2000. p. 193-200.

LITTLE, J. D. C. A proof for the queuing formula: L= ? W. Operations Research Journal, Baltimore, v. 9, n. 3, p. 383-387, May/June 1961.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of ACM, New York, v. 20, n. 1, p. 40-61, 1973.

LIU, J.; HONG, X. A traffic-aware energy efficient routing protocol for wireless sensor networks. In: INTERNATIONAL WORKSHOP ON COMPUTER ARCHITECTURE FOR MACHINE PERCEPTION AND SENSING, 2006, Montreal, Proceedings... Piscataway: IEEE, 2007. p. 142-147.

LIU, Y. et al. A hybrid interference model-based topology control algorithm. In: INTERNATIONAL CONFERENCE ON NETWORKED COMPUTING AND ADVANCED INFORMATION MANAGEMENT, 4., 2008. Gyeongju. Proceedings... Piscataway: IEEE2008. v. 1, p. 42-46.

LOZOYA, C.; MART, P.; VELASCO, M. In: MEDITERRANEAN CONFERENCE ON CONTROL AND AUTOMATION, 18., Marrakech, 2010. Proceedings... Piscataway: IEEE,2010. p. 267-272.

LU, C. et al. Rap: A real-time communication architecture for large-scale wireless sensor networks. In: REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, 8., 2002, San Jose. Proceedings... Los Alamitos: IEEE Computer Society, 2002. p. 55-66.

LUCK, R.; RAY, A. An observer-based compensator for distributed delays* 1. Automatica, New York, v. 26, n. 5, p. 903-908, 1990.

LYNCH, N. A. Distributed algorithms. San Francisco: Morgan Kaufmann, 1996.

MACÊDO, R. J. A. Failure detection in asynchronous distributed systems. In: WORKSHOP ON TESTS AND FAULT-TOLERANCE, 2., 2000. Curitiba. Proceedings... Porto Alegre: Sociedade Brasileira de Computação, 2000. p. 76-81.

MACÊDO, R. J. A. An integrated group communication infrastructure for hybrid real-time distributed systems. In: WORKSHOP ON REAL-TIME SYSTEMS, 9., 2007, Belém, Proceedings... Porto Alegre: Sociedade Brasileira de Computação, 2007. p. 81-88.

MACÊDO, R. J. A.; FREITAS, A. E. S. A generic group communication approach for hybrid distributed systems. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED APPLICATIONS AND INTEROPERABLE SYSTEMS, 9., 2009, Lisboa. Proceedings... Berlin: Springer-Verlag, 2009. p. 102-115.

MACÊDO, R. J. A.; GORENDER, S. Detectores perfeitos em sistemas distribuídos não síncronos. In: WORKSHOP DE TESTES E TOLERÂNCIA À FALHAS, 9., 2008, Rio de Janeiro. Anais... Rio de Janeiro: Sociedade Brasileira de Computação, 2008. p. 127-140.

MACÊDO, R. J. A.; GORENDER, S. Perfect failure detection in the partitioned synchronous distributed system model. In: INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY, 4., 2009, Fukuoka. Proceedings... Piscataway: IEEE Computer Society, 2009. p. 273-280.

MACÊDO, R. J. A.; GORENDER, S. Exploiting partitioned synchrony to implement accurate failure detectors. International Journal of Critical Computer-Based Systems, Inderscience Publishers, Inderscience Publishers, Geneva, v. 3, n. 3, p. 168-186, Nov. 2012.

MACÊDO, R. J. A.; LIMA, F. Improving the quality of service of failure detectors with SNMP and artificial neural networks. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004, Gramado. Anais... Porto Alegre: Sociedade Brasileira de Computação, 2004.

MAGALHÃES, A. P. Sistematizando o Desenvolvimento de Transformações Modelo a Modelo em uma Abordagem Dirigida a Modelos. Tese – Programa de Doutorado Multi-institucional em Ciência da Computação, Universidade Federal da Bahia, 2016.

MAGALHÃES, A. P.; ANDRADE, A.; MACIEL, R. S. A model driven transformation development process for model to model transformation. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 30, 2016, Maringá. Proceedings ... USA: ACM, 2016. p. 3–12.

MANJESHWAR, A.; AGRAWAL, D. P. Teen: A routing protocol for enhanced efficiency in wireless sensor networks. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM,15., 2000, San Francisco. Proceedings... [S.l.: s.n.], 2001. p. 30189a.

MANJESHWAR, A.; AGRAWAL, D. P. Teen: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In: IPDPS. IEEE Computer Society, 2002. p. 195.

MARTÍ, P. et al. Managing quality-of-control in network-based control systems by controller and message scheduling co-design. IEEE transactions on Industrial Electronics, Piscataway, v. 51, n. 6, p. 1159-1167, 2004.

MASTELLONE, S.; DORATO, P.; ABDALLAH, C. Finite-time stability for nonlinear networked control systems. In: MENINI; L.; ZACCARIAN, L.; ABDALLAH, C. T. (Ed.). Current trends in nonlinear systems and control: in honor of Petar Kokotovic and Turi Nicosia. [Berlin]: Springer 2006. p. 535-553.

MATHWORKS. Matlab: the language of technical computing. Nantick, USA, 2002.

MATHWORKS. Getting Started with Simulink. Nantick, USA, 2006.

MCNEILE, A.; ROUBTSOVA, E. Composition semantics for executable and evolvable behavioural modelling in MDA. In: WORKSHOP ON BEHAVIOUR MODELLING IN MODEL-DRIVEN ARCHITECTURE, 1., 2009, Enschede. Proceedings... New York: ACM, 2009.

MCQUEEN, B.; MCQUEEN, J. Intelligent transport: systems architectures. Boston: Artech House, 1999.

MDA. Model Driven Development (MDA Guide), 2003. Disponível em: <http://www.omg.org/mda/>. Acesso em: 01 Oct. 2003.

MELLOR, S. MDA Distilled: principles of model-driven architecture. Boston: Addisson-Wesley, 2004.

MENS, T. et al. Applying a model transformation taxonomy to graph transformation technology. Eletronic Notes in Theoretical Computer Science, [S.l.], v. 152, p. 143-159, 2006.

MERCER, C. W.; SAVAGE, S.; TOKUDA, H. Processor Capacity Reserves for Multimedia Operating Systems. Pittsburgh: School of Computer Science, Carnegie Mellon University, 1993.

MILLS, K. et al. An autonomic failure-detection algorithm. In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, 4., 2004, Redwood Shores. Proceedings... New York: ACM, 2004. p. 79-83.

MOSCIBRODA, T.; WATTENHOFER, R. Minimizing interference in ad hoc and sensor networks. In: WORKSHOP ON FOUNDATIONS OF MOBILE COMPUTING, 2005, Cologne. Proceedings... New York: ACM, 2005. p. 24-33.

MOSTEFAOUI, A.; MOURGAYA, E.; RAYNAL, M. Asynchronous implementation of failure detectors. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2003, San Francisco. Proceedings... [S.l.: s.n.], 2003.

MOTWANI, R. Approximation algorithms: book in preparation. [S.l.: s.n.], 1992.

MOYNE, J. R.; TILBURY, D. M. The emergence of industrial control networks for manufacturing control, diagnostics, and safety data. Proceedings of the IEEE, New York, v. 95, n. 1, p. 29-47, 2007.

MUNIZ, A.; ANDRADE, A.; LIMA, G. Integrating UML and UPPAAL for designing, specifying and verifying component-based real-time systems. Journal of Innovations in Systems and Software Engineering, [S.l.], p. 29-37, 2010.

MURACA, P.; PUGLIESE, P.; ROCCA, G. Extended kalman filtering using wireless sensor networks. In: INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 13., 2008, Hamburg. Proceedings... Piscataway: IEEE Xplore, 2008. p. 1084-1087.

MURRAY, R. M. et al. Future directions in control in an information-rich world. IEEE Control Systems Magazine, New York, v. 23, n. 2, p. 20-33, 2003.

NAGHSHTABRIZI, P.; HESPANHA, J. Designing an observer-based controller for a network control system. In: CONFERENCE ON DECISION AND CONTROL, 44., 2005, Sevilha. Proceedings... Piscataway: IEEE Operations Center, 2005. p. 848-853.

NAGHSHTABRIZI, P.; HESPANHA, J. P. Stability of networked control systems with variable sampling and delay. In: ANNUAL ALLERTON CONFERENCE ON COMMUNICATION, CONTROL, AND COMPUTING, 44., 2006, Monticello. Proceedings... Urbana, Ill.: Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 2006.

NI, S.-Y. et al. The broadcast storm problem in a mobile ad hoc network. ANNUAL ACM/IEEE INTERNATIONAL ARTICLE ON MOBILE COMPUTING AND NETWORKING, 5., 1999 Proceedings... New York: ACM, 1999. p. 151-162.

NILSSON, J. Real-time Control Systems with Delays. 1998. Thesis – Department of Automatic Control, Lund Institute of Technology, Lund, 1998.

NILSSON, J.; BERNHARDSSON, B.; WITTENMARK, B. Stochastic analysis and control of real-time systems with random time delays. Automatica, New York, v. 34, n. 1, p. 57-64, 1998.

NUNES, R. C.; JANSCH-PÔRTO, I. Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2004, Florence Proceeding... Los Alamitos: IEEE Computer Society, 2004. p. 753-761.

OGATA, K. Discrete-time control systems. 2nd. Englewood Cliffs: Prentice-Hall, 1995.

OGATA, K. Modern Control Engineering. 5th. Englewood Cliffs: Prentice-Hall, 2009.

OHTA, T.; INOUE, S.; KAKUDA, Y. An adaptive multihop clustering scheme for highly mobile ad hoc networks. In: INTERNATIONAL SYMPOSIUM ON AUTONOMOUS DECENTRALIZED SYSTEMS, 6., 2003, Pisa. Proceedings... Los Alamitos: IEEE Computer Society, 2003. p. 293-300.

OLIVEIRA, A. B. de. Uma Infraestrutura para reconfiguração dinâmica de escalonadores tempo real: modelos, algoritmos e aplicações. 2009. 95 f. Dissertação (Mestrado em Engenharia de Automação e Sistemas) – Universidade Federal de Santa Catarina, Florianópolis, 2009.

OLIVEIRA, R. G.; FARINES, J. M.; BECKER, L. B. Ferramenta para auxiliar o processo de verificação formal de propriedades em programas AADL. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEM ENGINEERING, 1., 2011, Florianópolis. Proceedings... Piscataway: IEEE, 2011.

PAPYRUS Modeling environment. 2014. Disponível em: <http://www.eclipse.org/papyrus/>. Acesso em: 31 Dec. 2014.

PARK, P.; ARAÚJO, J.; JOHANSSON, K. H. Wireless networked control system co-design. In: INTERNATIONAL CONFERENCE IN NETWORKING SENSING AND CONTROL, 2011, Delf. Proceedings...2011. Proceedings... Piscataway: NJ IEEE, 2011. p. 486-491.

PASSARINI, R. F.; FARINES, J. M.; BECKER, L. B. Solução para geração de modelos arquiteturais AADL a partir de modelos funcionais em simulink durante a concepção de sistemas embarcados. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 26., 2012, Natal. Anais... [Los Alamitos, Calif. : Conference Publishing Services, IEEE Computer Society], 2012.

PENG, C.; TIAN, Y. C. Networked h-1 control of linear systems with state quantization. Information Sciences, [New York], v. 177, p. 5763-5774, 2007.

PEREZ, D.; MORENO, U. F.; MONTEZ, C. B. Codesign of can networked control systems with remote controllers using jitter margin. In: ANNUAL CONFERENCE ON IEEE INDUSTRIAL ELECTRONICS, 32., 2006, Paris. Proceedings... Piscataway: NJ IEEE Service Center, 2006.

PETERSEN, S.; CARLSEN, S. Wireless HART versus ISA100.11a: The format war hits the factory floor. IEEE Industrial Electronics, [S.l.], v. 5, n. 4, 2011.

PLOPLYS, N. J.; KAWKA, P. A.; ALLEYNE, A. G. Closed-loop control over wireless networks. IEEE Control Systems Magazine, New York, v. 24, n. 3, p. 58-71, jun. 2004.

POLUSHIN, I. G.; LIU, P. X.; LUNG, C. H. On the model-based approach to nonlinear networked control systems. Automatica, New York, v. 44, n. 9, p. 2409-2414, 2008.

QVT. Query View Transformation (QVT). Disponível em: <http://www.omg.org/spec/QVT/1.1/>. Acesso em: 30 Dez. 2011.

RAJKUMAR, R. et al. Resource kernels: A resource-centric approach to real-time and multimedia systems. In: Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking. [S.l.: s.n.], 1998. p. 150-164.

RAMANATHAN, P.; HAMDAOUI, M. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. IEEE Transactions on Computers, Washington, v. 44, n. 12, p. 1443-1451, 1995.

RAPPAPORT, T. S. Wireless communications: principles & practice. Upper Saddle River: Prentice Hall, 1996.

RAY, A.; LIOU, L. W.; SHEN, J. H. State estimation using randomly delayed measurements. Journal of Dynamic Systems, Measurement, and Control, [New York], v. 115, p. 19-26, 1993.

RAYNAL, M. A short introduction to failure detectors for asynchronous distributed systems. ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) News, New York, v. 36, p. 53-70, Mar. 2005.

REZAYAT, P. et al. A novel real-time power aware routing protocol in wireless sensor networks. Journal of Computer Science, [S.l.], v. 10, n. 4, p. 300-305, 2010.

RICCOBENE, E.; SCANDURRA, P. Weaving executability into UML class models at PIM level. In: WORKSHOP ON BEHAVIOUR MODELLING IN MODEL-DRIVEN ARCHITECTURE, 1., 2009, Enschede. Proceedings... New York: ACM Digital Library, 2009.

RICKENBACH, P. v. et al. A robust interference model for wireless ad-hoc networks. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 19., 2005, Denver. Proceedings... Los Alamitos: IEEE Computer Society, 2005.

ROBOTA. Main Page - Robota. 2009. Disponível em: <http://robota.das.ufsc.br/>. Acesso em: 7 jun. 2009.

SÁ, A. S. de; MACÊDO, R. J. A. An adaptive failure detection approach for real-time distributed control systems over shared ethernet. In: INTERNATIONAL CONGRESS OF MECHANICAL ENGINEERING, 18., 2005, Ouro Preto. Proceedings... Rio de Janeiro: ABCM, 2006. v. 2.p. 43-50, 2006.

SÁ, A. S. de; MACÊDO, R. J. A. Um framework para prototipagem e simulação de detectores de defeitos para sistemas de tempo real distribuídos de controle e supervisão. In: WORKSHOP DE TESTES E TOLERÂNCIA A FALHAS, 8., 2007, Belém. Proceedings... [S.l.: s.n.], 2007. p. 43-56.

SÁ, A. S. de; MACÊDO, R. J. A. Detectores de defeitos autonômicos para sistemas distribuídos. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 28., 2010, Gramado, Brasil: Anais... Porto Alegre: SBC, 2010. p. 785-798.

SÁ, A. S. de; MACÊDO, R. J. A. QoS self-configuring failure detectors for distributed systems. In: ELIASSEN F., KAPITZA R. (Ed.). Distributed Applications and Interoperable Systems. Berlin: Springer-Verlag, 2010. p. 126-140. (Lecture Notes in Computer Science, v. 6115)

SÁ, A. S. de et al. Um procedimento para avaliação de redes ethernet comutada baseada em uma métrica de qualidade de controle. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 17., 2008. Anais... São Paulo: SBA, 2008. p. 1-6.

SAE. Architecture Analysis and Design Language (AADL – Behaviour Specification Annex). [S.l.], 2006.

SAGAR, C. et al. State/event-based software model checking. In: BOITEN, E. A.; DERRICK, J.; SMITH, G. In Integrated Formal Methods. Berlin: Springer-Verlag, 2004. p. 128-147.

SANTI, P. Topology control in wireless ad hoc and sensor networks. ACM Computing Survey, New York, v. 37, n. 2, p. 164-194, 2005.

SANTOS, T. L. M.; MORENO, U. F.; MONTEZ, C. B. Determination of a sampling period upper limit that guarantees closed loop stability in controller area network. In: WORKSHOP ON NETWORKED CONTROL SYSTEMS: TOLERANT TO FAULTS, 3., 2007, Nancy. Proceedings... [S.l.: s.n.], 2007.

SANTOS, T. L. M.; MORENO, U. F.; MONTEZ, C. B. Using the pole placement approach in a codesign procedure for networked controlled systems. IFAC Proceedings Volumes, Oxford, v. 40, n. 20, p. 154-159, 2007.

SATZGER, B. et al. A new adaptive accrual failure detector for dependable distributed systems. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2007, Seoul. Proceedings... New York: ACM, 2007. p. 551-555.

SCHMID, S.; WATTENHOFER, R. Algorithmic models for sensor networks. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 20., 2006, Rhodes Island. Proceedings... Los Alamitos: IEEE Computer Society, 2006.

SCHMIDT, D. Model-driven engineering. IEEE Computer, [S.l.], v. 39, n. 2, p. 25-31, February 2006.

SEILER, P.; SENGUPTA, R. An h8 approach to networked control. IEEE Transactions on Automatic Control, [S.l.], v. 50, n. 3, p. 356-364, 2005.

SELIC, B. The pragmatics of model-driven development. IEEE Software, Los Angeles, v. 20, n. 5, p. 19-25, Sept./Oct. 2003.

SENDALL, S.; KOZACZYNSKI, W. Model transformation: the heart and soul of model-driven software development. IEEE Software, Los Angeles, v. 20, n. 5, p. 42-45, 2003.

SHA, L. et al. Cyber-physical systems: A new frontier. In: TSAI, J. J. P.; YU, P. S. (Ed.). Machine Learning in Cyber Trust: security, privacy, and reliability. New York: Springer, 2009. p. 3-13.

SHAMMA, J. S.; ARSLAN, G. Dimensions of cooperative control. In:. Cooperative Control of Distributed Multi-Agent Systems. John Wiley & Sons, Ltd, 2007. cap. 1, p. 1-18.

SHEA, C. APROVE: a stable and robust VANET clustering scheme using affinity propagation. 2009. 107 f. Thesis – Department of Electrical and Computer Engineering, University of Toronto, 2009.

SIMULINK. Simulink Simulation and Model-Based Design. 2011. Disponível em: <http://www.mathworks.com/products/simulink/>. Acesso em: 1 Oct. 2011.

SINOPOLI, B. et al. Kalman filtering with intermittent observations. IEEE Transactions on Automatic Control, [S.l.], v. 49, n. 9, p. 1453-1464, 2004.

SO, K. C. W.; SIRER, E. G. Latency and bandwidth-minimizing failure detectors. In: EUROPEAN CONFERENCE ON COMPUTER SYSTEMS, 2., 2007, Lisbon. Proceedings... 2007. New York: ACM, 2007. p. 89-99.

SOARES, L. F. G.; BARBOSA, S. D. J. Programando em NCL 3.0 Versão 2.1. [S.l.]: PUC-Rio, 2011. Disponível em: <Http://www.telemidia.puc-rio.br/?q=pt-br/node/51>. Acesso em: 31 Dec. 2011.

STAHL, T.; VOLTER, M. Model-Driven Software Development: technology, engineering, management. Hoboken: John Wiley, 2006.

STEMMER, M. R. Redes locais e industriais: a integração da produção através das redes de comunicação. Santa Catarina: Editora UFSC, 2010.

STOJMENOVIC, I.; LIN, X. Gedir: loop-free location based routing in wireless networks. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 11., 1999. Proceedings... [S.l.: s.n.], 1999. p. 1025-1028.

TAHA, M. M. I.; HASAN, Y. M. Y. Vanet-dsrc protocol for reliable broadcasting of life safety messages. In: INTERNATIONAL SYMPOSIUM ON SIGNAL PROCESSING AND INFORMATION TECHNOLOGY, 2007. Proceedings... [S.l.: s.n.], 2007. p. 104-109.

TAI, A. T.; TSO, K. S.; SANDERS, W. H. Cluster-based failure detection service for large-scale ad hoc wireless network applications. INTERNATIONAL ARTICLE ON DEPENDABLE SYSTEMS AND NETWORKS, 2004, Florence. Proceedings... [S.l.: s.n.], 2004.

TANENBAUM, A. S. Computer networks. 4th. Upper Saddle River, USA: Prentice-Hall, 2003.

TANG, X.; YU, J. Networked control system: Survey and directions. In: LI K., FEI M., IRWIN G.W., MA S. (Ed.). Bio-Inspired Computational Intelligence and Applications. Berlin: Springer, 2007. p. 473-481. (Lecture Notes in Computer Science, v. 4688).

TISI, M.; CABOT, J.; JOUAULT, F. Improving higher-order transformations support in ATL. In: TRATT, L., GOGOLLA, M. (Ed.). Theory and practice of model transformations. Berlin: Springer, 2010. (Lecture Notes in Computer Science, v. 6142)

TORNGREN, M. et al. Tool supporting the co-design of control systems and their real-time implementation: Current status and future directions. In: CONFERENCE ON COMPUTER AIDED CONTROL SYSTEMS DESIGN; INTERNATIONAL CONFERENCE ON CONTROL APPLICATIONS; INTERNATIONAL SYMPOSIUM ON INTELLIGENT CONTROL, 2006. Proceedings.... [S.l.: s.n.], 2006. p. 1173-1180.

TRANSPORT SIMULATION SYSTEMS-TSS. AIMSUN Users Manual. Barcelona, 2000.

UML-MARTE. UML Profile for MARTE. [S.l.], 2008. <http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf>. Acesso em: 31 Dec. 2009.

VAGAS, A. INETMANET Framework for OMNEST/OMNeT++ 4.x (based on INET Framework). 2010. Disponível em: <http://wiki.github.com/inetmanet/inetmanet/>. Acesso em: 4 Oct. 2012.

VARGA, A. et al. The OMNeT++ discrete event simulation system. In: EUROPEAN SIMULATION MULTICONFERENCE, 2001, Prague. Proceedings... [S.l.: s.n.], 2001. p. 319-324.

VARRO, D.; PATARICZA, A. Automated formal verification of model transformations. In: WORKSHOP ON CRITICAL SYSTEMS DEVELOPMENT WITH UML. 2003. Proceedings... [S.l.: s.n], 2003. p. 63-78.

VERÍSSIMO, P.; CASIMIRO, A. The timely computing base model and architecture. IEEE Transactions on Computers, Washington, v. 51, n. 8, p. 916-930, August 2002.

VERÍSSIMO, P.; RODRIGUES, L. Distributed systems for systems architects. USA: Kluwer Academic Publishers, 2000.

VERÍSSIMO, P. E. Travelling through wormholes: a new look at distributed systems models. SIGACT News, New York, v. 37, p. 66-81, Mar. 2006.

WALSH, G. C.; BELDIMAN, O.; BUSHNELL, L. G. Asymptotic behaviour of nonlinear networked control systems. IEEE Transactions on Automatic Control, Notre Dame, USA, v. 46, p. 1093-1097, 1999.

WAN, P.-J. et al. Minimum-energy broadcasting in static ad hoc wireless networks. Wireless Networks, [S.l.], v. 8, p. 607-617, 2002.

WANG, Y.; LI, F. Vehicular ad hoc networks. In: MISRA, S.; WOUNGANG, I.; MISRA, S. C. (Ed.). Guide to Wireless Ad Hoc Networks. London: Springer, 2009. p. 503-525. (Computer Communications and Networks).

WANG, Z.; HO, D. W. C.; LIU, X. Robust filtering under randomly varying sensor delay with variance constraints. IEEE Transactions on Circuits and Systems Part 2: Express Briefs, Piscataway, v. 51, n. 6, p. 320-326, 2004.

WHITMORE, A.; AGARWAL, A.; XU, L. D. The internet of things: a survey of topics and trends. Information Systems Frontiers, v. 17, n. 2, p. 261-274, 2015.

WIESELTHIER, J. E.; NGUYEN, G. D.; EPHREMIDES, A. Energy-efficient broadcast and multicast trees in wireless networks. Mobile Network and Applications, Boston, v. 7, n. 6, p. 481-492, 2002.

WIESMANN, M.; URBAN, P.; DEFAGO, X. A SNMP based failure detection service. In: SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 25., 2006. Proceedings... Washington: IEEE Computer Society, 2006. p. 365-376.

WILLIG, A. Recent and emerging topics in wireless industrial communications: A selection. IEEE Transactions on Industrial Informatics, New York, v. 4, n. 2, p. 102-124, 2008.

WOLSEY, L. A. Integer Programming. [S.l.]: John Wiley & Sons, 1998.

WU, K. da; LIAO, W. Revisiting topology control for multi-hop wireless ad hoc networks. IEEE Transactions on Wireless Communications, New York, v. 7, n. 9, p. 3498-3506, 2008.

XIA, F.; SUN, Y.; TIAN, Y. C. Feedback scheduling of priority-driven control networks. Computer Standards & Interfaces, New York, v. 31, n. 3, p. 539-547, 2009.

XIE, L.; XIE, L. H. Stability analysis of networked sampled-data linear systems with markovian packet losses. IEEE Transactions on Automatic Control, [S.l.], v. 54, n. 6, p. 1368-1374, 2009.

XIONG, J.; LAM, J. Stabilization of linear systems over networks with bounded packet loss. Automatica, New York, v. 43, n. 1, p. 80-87, 2007.

XIONG, N. et al. On the quality of service of failure detectors based on control theory. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 20., 2006, Vienna. Proceedings... Los Alamitos: IEEE Computer Society, 2006. p. 75-80.

XU, K.; GERLA, M. A heterogeneous routing protocol based on a new stable clustering scheme. In: MILCOM, 2002, Anaheim. Proceedings... [S.l.: s.n.], 2002. v. 2, p. 838-843.

XU, Y.; HEIDEMANN, J.; ESTRIN, D. Geography-informed energy conservation for ad hoc routing. In: ANNUAL INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, 7., 2001, Rome. Proceedings...New York: ACM, 2001. p. 70-84.

XU, Y.; HESPANHA, J. P. Estimation under uncontrolled and controlled communications in networked control systems. In: CONFERENCE ON DECISION AND CONTROL, 44., 2005, Seville. Proceedings... Piscataway: IEEE Operations Center, 2005. p. 842-847.

YANG, F. et al. H-8 control for networked systems with random communication delays. IEEE Transactions on Automatic Control, [S.l.], v. 51, n. 3, p. 511-518, 2006.

YE, W.; HEIDEMANN, J.; ESTRIN, D. An energy-efficient mac protocol for wireless sensor networks. In: INFOCOM; ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES, 21., 2002, New York Hotel. Proceedings... Piscataway: IEEE, 2002.

YU, M. et al. Stabilization of networked control systems with data packet dropout and transmission delays: continuous-time case. European Journal of Control, [S.l.], v. 11, p. 40-49, 2005.

YU, Y.; GOVINDAN, R.; ESTRIN, D. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. 2001.

YUE, D.; HAN, Q.; LAM, J. Network-based robust H-∞ control of systems with uncertainty. Automatica, New York, v. 41, n. 6, p. 999-1007, 2005.

YUSUF, L.; CHESSEL, C.; GARDNER, D. Implement model driven development to increase the business value of your IT system. IBM DeveloperWorks Digital Library, [S.l.], 2006.

ZHANG, W.; BRANICKY, M. S.; PHILLIPS, S. M. Stability of networked control systems. IEEE Control Systems Magazine, New York, v. 21, n. 1, p. 84-99, 2001.

ZHAO, Y. B.; LIU, G. P.; REES, D. Integrated predictive control and scheduling co-design for networked control systems. Control Theory & Applications, Stevenage, v. 2, n. 1, p. 7-15, 2008.

Sobre os Autores

Aline Maria Santos Andrade. Graduação (UFBA), Mestrado (PUC-Rio) e Doutorado (PUC-Rio) em Ciência da Computação. Atualmente, é professora Associada do Departamento de Ciência da Computação da UFBA, pesquisadora do Laboratório de Sistemas Distribuídos (LaSiD/UFBA).

Alirio Santos de Sá. Bacharel em Ciência da Computação pela Faculdade Ruy Barbosa, Mestre em Mecatrônica e Doutor em Ciência da Computação, ambos realizados na Universidade Federal da Bahia (UFBA). É professor do Departamento de Ciência da Computação da UFBA, pesquisador do Laboratório de Sistemas Distribuídos (LaSiD/UFBA).

Ana Patrícia Fontes Magalhães Mascarenhas. Graduação em Informática (UCSAL), Mestrado em Mecatrônica (UFBA) e Doutorado em Ciência da Computação (UFBA).

Augusto Born de Oliveira. Graduado em Ciência da Computação e Mestre em Engenharia pela Universidade Federal de Santa Catarina. Atualmente é estudante de doutorado na University of Waterloo, Canadá.

Carlos Montez. É professor Associado da Universidade Federal de Santa Catarina. Graduou-se em Ciência da Computação pela UFRJ. Completou seu mestrado em Ciências da Computação pela UFSC em 1995 e doutorado em Engenharia Elétrica pela UFSC em Dezembro de 1999.

Eduardo Cambruzzi. É professor do Instituto Federal de Educação, Ciência e Tecnologia da Bahia – Campus Valença. É graduado em Ciência da Computação, Mestre em Engenharia Elétrica e Doutor em Automação e Sistemas pela Universidade Federal de Santa Catarina.

Eduardo Camponogara. Doutor em Engenharia Elétrica e em Ciência da Computação pela Carnegie Mellon University (2000). É professor no Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina.

Flávio Assis. Bacharel (UFMG), Mestre (Unicamp) e Doutor (Universidade Técnica de Berlim, Alemanha) em Ciência da Computação. É Professor Associado no Departamento de Ciência da Computação da UFBA (Universidade Federal da Bahia) e pesquisador do Laboratório de Sistemas Distribuídos (LaSiD/UFBA).

George Lima. Doutor (Universidade de York, 2003), Mestre (Unicamp, 1996) e Bacharel (UFBA, 1993) em Ciência da Computação. É Professor Associado do Departamento de Ciência da Computação da UFBA (Universidade Federal da Bahia) e pesquisador do Laboratório de Sistemas Distribuídos (LaSiD/UFBA).

Jean-Marie Farines. Possui doutorado no Institut National Polytechnique de Toulouse (INPT, 1979). É Professor Titular do Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina (DAS-UFSC).

Leandro Buss Becker. É Bacharel em Informática pela Universidade Federal de Santa Maria e Mestre e Doutor em Computação pela Universidade Federal do Rio Grande do Sul. É Professor no Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina.

Raimundo José de Araújo Macêdo. Graduação, Mestrado e Doutorado em Ciência da Computação, realizados na UFBA, Unicamp e University of Newcastle upon Tyne - UK, respectivamente. É professor Titular do Departamento de Ciência da Computação da UFBA, coordenador do Laboratório de Sistemas Distribuídos (LaSiD/UFBA).

Tanisia de Carli Folleto. Graduada em Matemática pela Universidade Federal de Santa Maria (2002), Mestre em Modelagem Matemática pela Universidade Regional do Noroeste do Estado do Rio Grande do Sul (2008) e Doutora em Engenharia de Automação e Sistemas pela Universidade Federal de Santa Catarina (2013). É Professora no Instituto Federal Farroupilha- Campus Júlio de Castilhos - RS.

Ubirajara F. Moreno. Possui graduação em Engenharia Industrial Elétrica pela Universidade Tecnológica Federal do Paraná (1994), mestrado em Engenharia Elétrica pela Universidade Federal de Santa Catarina (1997) e doutorado em Engenharia Elétrica pela Universidade Estadual de Campinas (2001). É professor da Universidade Federal de Santa Catarina.

Formato: 16,5 x 24,0
Papel: Alta Alvura 75 g/m² (miolo)
Impressão: EDUFBA
Capa e acabamento: I.Bigraf
Tiragem: 500 exemplares

Este livro traz resultados de pesquisas para desafios fundamentais na construção dos sistemas industriais distribuídos modernos, cobrindo aspectos de controle distribuído e via rede, tolerância a falhas, escalonamento de processos, redes de sensores sem fio e projeto de software. Os assuntos cobertos podem ser utilizados em disciplinas de pós-graduação e disciplinas avançadas da graduação, em programas de ciência da computação, mecatrônica, engenharia de controle e automação, engenharia elétrica, entre outras áreas pertinentes. A abordagem multi-interdisciplinar, centrada em problemas fundamentais da computação e comunicação, ao invés de tecnologias específicas, distingue esta obra de outras dedicadas aos sistemas industriais.

ISBN 978-85-232-1675-7
9 788523 216757