ROVERLI PEREIRA ZIWICH

ESTRATÉGIAS EFICIENTES PARA IDENTIFICAÇÃO DE FALHAS UTILIZANDO O DIAGNÓSTICO BASEADO EM COMPARAÇÕES

CURITIBA

2013

ROVERLI PEREIRA ZIWICH

ESTRATÉGIAS EFICIENTES PARA IDENTIFICAÇÃO DE FALHAS UTILIZANDO O DIAGNÓSTICO BASEADO EM COMPARAÇÕES

Tese apresentada ao Programa de Pós-Graduação em Informática do Setor de Ciências Exatas da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.
Orientador: Prof. Elias Procópio Duarte Jr.

CURITIBA

2013

Ziwich, Roverli Pereira
Estratégias eficientes para identificação de falhas utilizando o diagnóstico baseado em comparações / Roverli Pereira Ziwich. – Curitiba, 2013.
148 f. : il. ; graf., tab.

Tese (doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Elias Procópio Duarte Jr.

1. Analise de sistemas – Desempenho. I. Duarte Junior, Elias Procópio. II. Título.

CDD 004.24

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Roverli Pereira Ziwich, avaliamos a tese de doutorado intitulada *"Estratégias Eficientes para Identificação de Falhas Utilizando o Diagnóstico Baseado em Comparações"*, cuja defesa pública foi realizada no dia 12 de abril de 2013, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela **aprovação** do candidato.

Curitiba, 12 de abril de 2013.

Prof. Dr. Elias Procópio Duarte Júnior
DINF/UFPR – Orientador

Profa. Dra. Jussara Marques de Almeida
UFMG – Membro Externo

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Prof. Dr. Altair Olivo Santin
PUC/PR – Membro Externo

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Membro Interno

Prof. Dr. Luiz Carlos Pessoa Albini
DINF/UFPR – Membro Interno

AGRADECIMENTOS

Gostaria de agradecer a todos que foram fundamentais neste doutorado; um trabalho desta natureza nunca é realizado sozinho. A Deus primeiramente, por sua indiscutível importância; sem Ele nada seria possível. Ao Prof. Elias Procópio Duarte Jr. por ser um excepcional orientador, além de um grande amigo. À minha família, minha esposa Myrna Okamoto Ziwich e meu filho Vinicius Okamoto Ziwich, por todo suporte, carinho e amor; e aos meus pais, Roberto Ziwich e Vera Lucia Ziwich, por estarem sempre presentes. Agradeço aos colegas Emanuel Amaral Schimidt, Renan Miranda, Luis Fernando Machado e Glaucio Pessoa da Silveira pelos trabalhos conjuntos com o simulador Fireflies. Agradeço também aos professores Luís Carlos Erpen de Bona, Luiz Carlos Pessoa Albini, Jussara Marques de Almeida e Altair Olivo Santin pelas participações nas bancas e pelos precisos apontamentos para a revisão deste trabalho. Agradeço também à Universidade Federal do Paraná, lugar do qual já me considero parte. Obrigado ainda aos professores e à secretaria do Departamento de Informática, pelas excelentes disciplinas e pelo suporte dado sempre que precisei. Por fim, agradeço também a todos que de alguma forma contribuíram para a realização deste trabalho.

Pelo esforço é que se abre o caminho.

Virgílio

RESUMO

O diagnóstico baseado em comparações é uma forma realista para detectar falhas em *hardware*, software, redes e sistemas distribuídos. O diagnóstico se baseia na comparação de resultados de tarefas produzidos por pares de unidades para determinar quais são as unidades falhas e sem-falha do sistema. Qualquer diferença no resultado da comparação indica que uma ou ambas as unidades estão falhas. O diagnóstico completo do sistema é baseado no resultado de todas as comparações. Este trabalho apresenta um novo algoritmo de diagnóstico para identificar falhas em sistemas de topologia arbitrária com base no modelo MM*. A complexidade do algoritmo proposto é $O(t^2\Delta N)$ no pior caso para sistemas de N unidades, onde t denota o número máximo permitido de unidades falhas e Δ é o grau da unidade de maior grau no sistema. Esta complexidade é significativamente menor que a dos outros algoritmos previamente publicados. Além da especificação do algoritmo e das provas de correção, resultados obtidos através da execução exaustiva de experimentos são apresentados, mostrando o desempenho médio do algoritmo para diferentes sistemas. Além do novo algoritmo para sistemas de topologia arbitrária, este trabalho também apresenta duas outras soluções para detecção e combate à poluição de conteúdo, ou alterações não autorizadas, em transmissões de mídia contínua ao vivo em redes P2P – a primeira é uma solução centralizada e que realiza o diagnóstico da poluição na rede, e a segunda é uma solução completamente distribuída e descentralizada que tem o objetivo de combater a propagação da poluição na rede. Ambas as soluções utilizam o diagnóstico baseado em comparações para detectar alterações no conteúdo dos dados transmitidos. As soluções foram implementadas no Fireflies, um protocolo escalável para redes *overlay*, e diversos experimentos através de simulação foram conduzidos. Os resultados mostram que ambas as estratégias são soluções viáveis para identificar e combater a poluição de conteúdo em transmissões ao vivo e que adicionam baixa sobrecarga ao tráfego da rede. Em particular a estratégia de combate à poluição foi capaz de reduzir consideravelmente a poluição de conteúdo em diversas configurações, em vários casos chegando a eliminá-la no decorrer das transmissões.

ABSTRACT

Comparison-based diagnosis is a practical approach to detect faults in hardware, software, and network-based systems. Diagnosis is based on the comparison of task outputs returned by pairs of system units in order to determine whether those units are faulty or fault-free. If the comparison results in a mismatch then one ore both units are faulty. System diagnosis is based on the complete set of all comparison results. This work introduces a novel diagnosis algorithm to identify faults in t-diagnosable systems of arbitrary topology under the MM* model. The complexity of the proposed algorithm is $O(t^2\Delta N)$ in the worst case for systems with N units, where t denotes the maximum number of faulty units allowed and Δ corresponds to the maximum degree of a unit in the system. This complexity is significantly lower than those of previously published algorithms. Besides the algorithm specification and correctness proofs, exhaustive simulations results are presented, showing the typical performance of the algorithm for different systems. Moreover, this work also presents two different strategies to detect and fight content pollution in P2P live streaming transmissions – the first strategy is centralized and performs the diagnosis of content pollution in the network, and the second strategy is a completely distributed solution to combat the propagation of the pollution. Both strategies employ comparison-based diagnosis in order to detect any modification in the data transmitted. The solutions were also implemented in Fireflies, a scalable and fault-tolerant overlay network protocol, and a large number of simulation experiments were conduced. Results show that both strategies are feasible solutions to identify and fight content pollution in live streaming sessions and that they add low overhead in terms of network bandwidth usage. In particular, the solution proposed to combat content pollution was able to significantly reduce the pollution over the system in diverse network configurations – in many cases the solution nearly eliminated the pollution during the transmission.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Como atualmente está na casa de bilhões o número estimado de pessoas que utilizam a Internet [111], o bom funcionamento da rede e dos sistemas que nela executam é cada vez mais importante para indivíduos e organizações. Por outro lado, ataques e ações de vandalismo têm se tornado cada vez mais comuns [180, 27, 153]. Além do crescimento da utilização da Internet, também é fato que os sistemas computacionais estão cada vez maiores e mais complexos: processadores já combinam centenas de núcleos em um único *chip*, redes conectam centenas de milhares de unidades, softwares executam de forma distribuída em diversos dispositivos computacionais [116, 90, 175]. É muito provável que unidades destes sistemas irão, em algum momento, deixar de funcionar corretamente, produzindo resultados diferentes do especificado. O objetivo do diagnóstico em nível de sistema é justamente identificar quais unidades estão funcionando corretamente e quais estão falhas [158, 176]. Se a falha de uma ou mais unidades causar a falha completa do sistema, usuários podem ser fortemente prejudicados. Com isso, a preocupação com a monitoração de sistemas, visando a detecção de ataques, violações ou simplesmente comportamento anormal, têm crescido constantemente [107, 106, 205, 58].

O diagnóstico em nível de sistema baseado em comparações [59] é uma forma realista para a identificação de falhas em sistemas computacionais. Este paradigma de diagnóstico é uma teoria que tem sido estudada há mais de três décadas e diversas aplicações têm sido apresentadas ao longo destes anos.

Aplicações tradicionais incluem a detecção de falhas em redes de múltiplos computadores como, por exemplo, redes de multiprocessadores interconectados [107, 102, 106, 205, 37, 198]. Em [193] o diagnóstico de falhas baseado em comparações é aplicado a projetos de software. Em ambas os casos, existe a necessidade de realização de testes e

diagnóstico de forma eficiente. Em especial, o diagnóstico baseado em comparações tem se mostrado interessante para os sistemas multiprocessados [187] e já tem sido aplicado para o diagnóstico de falhas em circuitos integrados complexos [164, 88, 166]. Além disso, outras novas aplicações têm aparecido recentemente, e incluem: identificação de falhas em redes ad hoc [73, 34] e em redes neurais artificiais [76, 65, 77]; verificação de integridade em informações distribuídas e replicadas [208], espalhadas por uma rede como, por exemplo, a Internet; e, a verificação da manipulação de resultados de processos por unidades maliciosas em plataformas de computação de grade [146, 147, 145, 144].

Na sequência, este capítulo está dividido em 4 seções e apresenta primeiramente uma breve visão sobre o diagnóstico baseado em comparações. A Seção 1.2 introduz o novo algoritmo de diagnóstico baseado em comparações proposto para sistemas de topologia arbitrária. A seguir, a Seção 1.3 descreve as duas soluções propostas para o diagnóstico e combate à poluição de conteúdo em transmissões ao vivo em redes P2P. Por fim, na Seção 1.4, a organização deste trabalho é apresentada.

1.1 Diagnóstico Baseado em Comparações

O diagnóstico baseado em comparações utiliza a comparação do resultado de tarefas produzidos por pares de unidades para determinar quais são as unidades sem-falha e quais são as unidades falhas do sistema. O conjunto com os resultados de todas as comparações é chamado de síndrome do sistema. Os primeiros modelos de diagnóstico baseados em comparações foram propostos por Malek [143] e em seguida por Chwa e Hakimi [42]. Nestes modelos, é assumido que em um sistema de N unidades, a comparação dos resultados produzidos pela saída da execução de tarefas de alguns ou todos os pares de unidades é possível. Qualquer diferença na comparação indica que uma ou ambas as unidades estão falhas. Estes modelos assumem a existência de um observador central. O observador central armazena as informações das saídas de tarefas e através das comparações das saídas obtém o diagnóstico completo do sistema, ou seja, identifica quais são as unidades

falhas do sistema. A diferença deste modelo proposto por Chwa e Hakimi para o modelo proposto por Malek, é que no modelo de Chwa e Hakimi duas unidades falhas, quando submetidas à mesma tarefa, podem produzir as mesmas saídas, ou seja, a comparação destas saídas pode resultar em igualdade.

O modelo MM – proposto por Maeng e Malek em [140] – é uma extensão do modelo de diagnóstico baseado em comparações proposto inicialmente por Malek [143]. O modelo MM permite que as próprias unidades do sistema realizem a comparação das saídas das tarefas, e então os resultados das comparações são enviados a um observador central que realiza o diagnóstico completo do sistema. A única restrição é que a unidade que realiza a comparação deve ser diferente das duas unidades que executam as tarefas. Maeng e Malek também em [140] apresentam um caso especial do modelo MM, chamado MM*, no qual uma unidade testadora executa comparações para todas as suas unidades vizinhas, em pares.

Sengupta e Dahbura em [169] propõem uma generalização do modelo MM onde a própria unidade testadora pode ser uma das unidades que são comparadas, e além disso também apresentam as condições necessárias para um sistema ser diagnosticável com base no modelo MM. Outros modelos de diagnóstico baseado em comparações que são completamente distribuídos, ou seja, não assumem a existência de um observador central são apresentados em [5, 208], nos quais as próprias unidades sem-falha testam e classificam as unidades do sistema em conjuntos.

Dois algoritmos de diagnóstico foram previamente propostos para sistemas de topologia arbitrária com o objetivo de determinar quais são as unidades falhas a partir da síndrome do sistema. Sengupta e Dahbura em [169] apresentam um algoritmo de diagnóstico de ordem de complexidade $O(N^5)$ para identificar as unidades falhas com base no modelo MM*, onde N é o número de unidades do sistema. Mais recentemente, também para o modelo MM*, um algoritmo de diagnóstico de ordem de complexidade $O(N\Delta^3\delta)$ – onde Δ e δ são respectivamente os graus das unidades de maior e menor grau do sistema – foi apresentado por Yang e Tang em [198]. Entretanto, é importante notar que

para sistemas completamente conectados (onde $\Delta = \delta = N - 1$) ambos os algoritmos de Sengupta e Dahbura e de Yang e Tang são $O(N^5)$.

1.2 Um Novo Algoritmo $O(t^2\Delta N)$ de Diagnóstico Baseado em Comparações para Sistemas de Topologia Arbitrária

Este trabalho apresenta um novo algoritmo de diagnóstico para a identificação das unidades falhas em sistemas de topologia arbitrária diagnosticáveis com base no modelo MM*. A complexidade do algoritmo proposto é $O(\Delta N^2)$ quando $t^2 < N$ ou $O(t^2\Delta N)$ no caso contrário, ou seja, o algoritmo é $O(t^2\Delta N)$ no pior caso, onde t denota o número máximo permitido de unidades falhas em um sistema de N unidades e Δ é o grau da unidade de maior grau no sistema. Este algoritmo portanto apresenta uma complexidade que é significativamente menor do que a dos dois outros algoritmos propostos por Sengupta e Dahbura e por Yang e Tang.

Como a síndrome do sistema tem tamanho $O(\Delta^2 N)$, a complexidade da estratégia proposta é muito próxima da complexidade de se percorrer os elementos da síndrome uma única vez. Além disso, considerando sistemas completamente conectados onde $\Delta = \delta = N - 1$, a complexidade do algoritmo proposto neste trabalho é $O(N^3)$ quando $t^2 < N$ ou $O(t^2N^2)$ no caso contrário.

A solução proposta é também a primeira que realiza o diagnóstico de sistemas de topologia arbitrária com base apenas na síndrome de comparações. Ambos os algoritmos previamente publicados [198, 169] aplicam técnicas de diagnóstico em nível de sistema e convertem a síndrome de comparações para uma síndrome de testes [158], para que o diagnóstico seja realizado.

Além da especificação do algoritmo e das provas de correção, resultados obtidos através da execução de experimentos também são apresentados neste trabalho. Os resultados mostram que o número de testes executados pelo algoritmo proposto é em média cerca de $N^{2.5}$ testes. Além disso, os resultados mostram que, para os maiores sistemas experimen-

tados, a parte do algoritmo proposto que possui o pior caso da ordem de complexidade geralmente não é executada. Em outras palavras, para os maiores sistemas, em mais de 96% dos experimentos apenas a parte do algoritmo que possui complexidade $O(\Delta N^2)$ é executada.

1.3 Detecção e Combate à Poluição de Conteúdo em Transmissões ao Vivo em Redes P2P

Transmissões ao vivo, notadamente de vídeos, estão se tornando cada vez mais populares na Internet [135] e diversos sistemas para transmissões ao vivo que utilizam redes P2P surgiram nos últimos anos – como por exemplo o PPLive[1], o SopCast[2], e o PPStream[3] – e sustentam milhões de usuários registrados [99]. As redes P2P possuem algumas vantagens sobre o formato tradicional cliente-servidor para transmissões ao vivo pois os próprios usuários podem compartilhar o conteúdo que é transmitido. Desta forma a quantidade, capacidade de processamento e de largura de banda dos servidores que distribuem o conteúdo que é transmitido nas redes P2P pode ser significativamente menor do que a dos mesmos servidores nas redes que utilizam o formato tradicional.

Nos sistemas P2P, a geração do conteúdo que é transmitido é realizada por um *servidor fonte*. O conteúdo transmitido é dividido em pequenos pedaços, chamados *chunks*. O servidor fonte é responsável pela difusão inicial dos *chunks* na rede que, por sua vez, são compartilhados entre os usuários – os *peers* – da rede.

Por outro lado, como os próprios *peers* são responsáveis por disseminar o conteúdo transmitido, a poluição de conteúdo nas transmissões ao vivo em redes P2P é um dos desafios que continuam relevantes [99]. Um ataque de poluição de conteúdo consiste na modificação não autorizada dos dados (ou *chunks*) transmitidos. A modificação dos *chunks* pode ser de diferentes tipos [91, 53, 135], que incluem: a troca de conteúdo; a

[1]PPLive - *http://www.pplive.com/en*
[2]SopCast - *http://www.sopcast.com*
[3]PPStream - *http://www.ppstream.com*

criação de novos dados; e até a destruição ou omissão de *chunks*.

Outras características que agravam o problema da poluição de conteúdo nas transmissões ao vivo incluem o limite de tempo no qual o conteúdo transmitido possui para alcançar os *peers* da rede, e o *churn*, isto é, o fato de *peers* entrarem e saírem da rede continuamente durante a transmissão. Estas características são relevantes pois a detecção de conteúdo poluído e a consequente criação de novas solicitações pode causar atrasos, e até saltos, na transmissão assistida pelos usuário [196, 203].

Algumas soluções que tratam o problema da poluição de conteúdo em transmissões ao vivo assumem que todos os *peers* sabem previamente, ou recebem durante a própria transmissão o valor *hash* dos respectivos *chunks* [191]. Esta estratégia é bastante usada para tratar falhas físicas nos canais de comunicação, mas ainda permite a um *peer* malicioso modificar indevidamente um *chunk* e retransmití-lo juntamente com um novo valor *hash* correspondente.

Outras soluções ainda propõem o uso de assinaturas digitais, ou seja, criptografia de chave pública, para todos os *chunks* que são transmitidos [97]. A assinatura digital é gerada pelo servidor fonte e transmitida juntamente com os *chunks* pela rede. Nesta estratégia, cada *peer* que recebe um *chunk* deve conferir se a assinatura digital é válida. Por outro lado este é um procedimento que pode ser considerado computacionalmente custoso, dependendo dos dispositivos usados pelos usuários da transmissão [53].

Este trabalho apresenta duas soluções que utilizam o diagnóstico baseado em comparações para detectar e combater poluição de conteúdo em transmissões de mídia contínua ao vivo em redes P2P. Diferente das anteriores, as soluções propostas neste trabalho não utilizam criptografia de chave pública e não utilizam o envio de valores *hash* junto à transmissão. A primeira solução proposta é baseada em um *tracker* central e realiza o diagnóstico (apenas a detecção) da poluição na rede; já a segunda solução é distribuída e descentralizada e é voltada ao combate da propagação da poluição em transmissões ao vivo em redes *overlay*.

Ambas as soluções utilizam o diagnóstico baseado em comparações [59] para detec-

tar alterações no conteúdo dos dados transmitidos. Cada *peer* do sistema executa comparações periódicas sobre determinados *chunks* de seus vizinhos. Com base no resultado das comparações executadas por todos os *peers*, a primeira solução realiza uma classificação unificada de todos os *peers* com o objetivo de determinar se há poluição de dados. Já na segunda solução, com base nos resultados das comparações, cada *peer*, de forma independente dos demais, deixa de solicitar *chunks* aos seus vizinhos considerados poluidores.

É importante destacar que os algoritmos empregados nestas duas soluções de diagnóstico e combate à poluição de conteúdo são diferentes do algoritmo proposto – introduzido na seção anterior – para o diagnóstico de falhas em sistemas de topologia arbitrária com base no modelo MM*. Em transmissões ao vivo este algoritmo não pode ser aplicado pois o modelo MM* assume que a comparação de tarefas executadas por duas unidades falhas resulta em diferença. Por outro lado, em transmissões ao vivo, a comparação de dois *chunks* poluídos, retornados por diferentes *peers*, pode resultar em igualdade.

As duas soluções propostas foram implementadas no Fireflies [114], um protocolo escalável para redes *overlay*. O Fireflies usa a estratégia *pull-based* para a transmissão dos dados e emprega a topologia *mesh* [100]. A implementação foi realizada usando o mesmo simulador Fireflies descrito em [97]. Os resultados experimentais mostram que ambas as estratégias são soluções viáveis para identificar e combater a poluição de conteúdo em transmissões ao vivo e adicionam baixa sobrecarga ao tráfego da rede. Em particular sobre a estratégia de combate à poluição, em diversas configurações a solução foi capaz de reduzir consideravelmente a poluição de conteúdo, em vários casos chegando a eliminá-la no decorrer das transmissões.

1.4 Organização deste Trabalho

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o diagnóstico em nível de sistema baseado em comparações: em particular os modelos MM

e MM* de diagnóstico baseado em comparações, os algoritmos de diagnóstico baseado em comparações para sistemas de topologia arbitrária de Sengupta e Dahbura e de Yang e Tang, e também os modelos generalizados e distribuídos baseados em comparações. O Capítulo 3 apresenta o novo algoritmo $O(t^2\Delta N)$ de diagnóstico baseado em comparações proposto para sistemas de topologia arbitrária. Já o Capítulo 4 apresenta as duas soluções propostas para o diagnóstico e combate à poluição de conteúdo em transmissões ao vivo em redes P2P. O Capítulo 5 traz as conclusões e trabalhos futuros, e na sequência, segue um registro das publicações realizadas neste doutorado.

Este trabalho ainda possui dois apêndices. O Apêndice A apresenta em detalhes diversas outras abordagens para o diagnóstico baseado em comparações. Por fim, o Apêndice B traz uma lista dos termos, abreviações e definições mais importantes que aparecem neste trabalho.

CAPÍTULO 2

DIAGNÓSTICO EM NÍVEL DE SISTEMA BASEADO EM COMPARAÇÕES

Como falhas são inevitáveis nos sistemas computacionais, é importante ser capaz de determinar quais unidades do sistema estão funcionando e quais unidades estão falhas. O diagnóstico baseado em comparações [59] é uma forma realista para a identificação das unidades falhas nestes sistemas.

Este capítulo está dividido em 5 seções e apresenta, primeiramente, o diagnóstico em nível de sistema. Na sequência são apresentados os primeiros modelos de diagnóstico baseado em comparações, os modelos MM e MM*, os algoritmos de tempo polinomial para sistemas de topologia arbitrária para o modelo MM*, e, por fim, os modelos generalizados e distribuídos baseados em comparações.

É importante ressaltar que o Apêndice A apresenta – baseado em [59], além de trabalhos mais recentes – diversas outras abordagens para o diagnóstico baseado em comparações, que incluem, entre outros, os modelos probabilísticos baseado em comparações e o modelo baseado em *broadcast* confiável, e resultados para a diagnosticabilidade de uma série de topologias de interconexão de redes como: hipercubos, *enhanced hypercubes*, redes borboletas, *locally twisted cubes*, *hypercube-like networks*, grafos estrelas, *matching composition networks*, redes t-conectadas e redes produto.

2.1 Diagnóstico em Nível de Sistema

O problema da identificação de unidades falhas em sistemas computacionais complexos continua sendo um dos problemas de grande relevância. Devido à crescente complexidade dos sistemas computacionais e constante aumento de tamanho, em algum momento uni-

dades destes sistemas irão falhar, produzindo resultados diferentes dos especificados. Para que usuários desses sistemas não sejam afetados, é importante ser capaz de determinar se existem e quais são as unidades falhas destes sistemas. Este problema é conhecido como o problema do *diagnóstico em nível de sistema*, e o primeiro modelo proposto para o diagnóstico em nível de sistema foi o modelo PMC [158], nomeado com as iniciais dos autores: Preparata, Metze e Chien.

O modelo PMC assume um sistema S que consiste de um conjunto de N unidades independentes, u_0, u_1, ..., u_{N-1}. Alternativamente, neste trabalho uma unidade u_i será também referenciada por *unidade* i, *nodo* i, ou mesmo *processador* i. Assume-se que cada unidade u_i sempre está em um de dois estados, *falha* ou *sem-falha*. O modelo PMC assume ainda que o sistema é completamente conectado (*fully connected*), ou seja, cada unidade do sistema é conectada a todas as demais.

O diagnóstico é baseado na habilidade das unidades testarem o estado de outras unidades do sistema [148, 113]. Uma unidade é testada como um todo, não é possível testar parte de uma unidade, e o estado de uma unidade não muda durante a realização do diagnóstico. No modelo PMC, um teste envolve a aplicação controlada de estímulos e a observação da resposta correspondente retornada pela unidade testada. Preparata, Metze e Chien definem um teste como um "procedimento de diagnóstico" específico e personalizado para cada sistema. Uma unidade que realiza um teste também é chamada de unidade testadora.

O modelo PMC assume que uma unidade sem-falha sempre executa os testes de forma correta, isto é, uma unidade testadora sem-falha sempre pode determinar se uma unidade testada está falha ou sem-falha. Mais precisamente, com base no resultado dos estímulos aplicados, o resultado do teste é classificado como *pass* (0) ou *fail* (1). Entretanto, os autores enfatizam que informações mais detalhadas sobre a falha podem ser retidas para futuras investigações [158].

Enquanto assume-se que unidades sem-falha são capazes de executar os testes corretamente, nenhuma asserção é feita sobre os testes executados pelas unidades falhas, ou

seja, as unidades falhas podem produzir resultados incorretos para os testes [158, 95]. Ao conjunto de todos os possíveis resultados para um teste dá-se o nome de regras de invalidação (*invalidation rules*) e a Tabela 2.1 mostra as regras de invalidação para o modelo PMC.

Unidade Testadora	**Unidade Testada**	**Resultado**
sem-falha	sem-falha	0 (*pass*)
sem-falha	falha	1 (*fail*)
falha	sem-falha	0 ou 1
falha	falha	0 ou 1

Tabela 2.1: Regras de invalidação para o modelo PMC.

O conjunto dos testes que são realizados é chamado de *configuração de testes* (ou *connection assignment*), e ao conjunto com os resultados de todos os testes dá-se o nome de *síndrome* do sistema – no modelo PMC, a síndrome do sistema também pode ser chamada de *síndrome de testes*. A síndrome é processada por uma entidade externa, chamada de *observador central*, que realiza o diagnóstico do sistema, isto é, determina o estado de todas as unidades do sistema.

O modelo PMC define um grafo direcionado para representar a configuração de testes. Os vértices deste grafo são as unidades do sistema, e existe uma aresta direcionada da unidade i para a unidade j se a unidade i testa a unidade j. Cada aresta (i, j) possui um rótulo (*label*) com o resultado do teste, $a_{i,j} = \{0, 1\}$. Se uma unidade i testa uma unidade j com falha, então $a_{i,j} = 1$; por outro lado, se a unidade i testa a unidade j como sem-falha, então $a_{i,j} = 0$. Estes resultados são válidos apenas se a unidade testadora é sem-falha, caso contrário o resultado dos testes não são confiáveis.

Como um exemplo, a Figura 2.1 mostra um sistema com 5 unidades sequencialmente identificadas, onde somente a unidade 1 é falha. Cada unidade executa um teste sobre a próxima unidade considerando uma disposição circular, ou seja, a unidade i testa a unidade $(i \bmod N) + 1$. O rótulo de cada aresta direcionada mostra o resultado do teste $a_{i,j}$ correspondente. A síndrome pode ser representada por um vetor de 5 *bits*, e neste exemplo é $(x, 0, 0, 0, 1)$, onde x pode ser 0 ou 1. Com base na síndrome, a unidade 5

corretamente identifica que a unidade 1 é falha; as unidades 2, 3 e 4 também identificam corretamente, como sem-falha, o estado das unidades respectivamente testadas por elas.

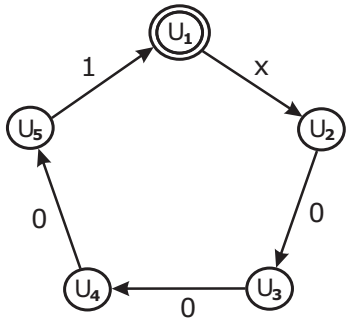


Figura 2.1: Um sistema com 5 unidades onde a unidade 1 é falha.

Dependendo da quantidade de unidades falhas e da configuração dos testes, é impossível realizar corretamente o diagnóstico do sistema [158]. Um sistema é chamado de *t-diagnosticável em um passo* (*one-step t-diagnosable*) se toda unidade falha do sistema puder ser identificada desde que o número de unidades falhas não seja maior que t. Além disso, um sistema é definido com *sequencialmente t-diagnosticável* (*sequentially t-diagnosable*) se pelo menos uma unidade puder ser identificada e ser reparada ou substituída, assim os testes podem continuar já considerando a unidade reparada, com o objetivo de diagnosticar, em algum momento, todas as unidades falhas do sistema.

Em um sistema t-diagnosticável o problema de determinar o valor máximo para t é chamado de o *problema da diagnosticabilidade* (ou *diagnosability problem*). Preparata, Metze e Chien apresentam as condições necessárias para a t-diagnosticabilidade do modelo PMC [158]. Posteriormente os autores de [95] caracterizam o modelo PMC e provam que as seguintes condições são, além das condições necessárias, as condições suficientes para um sistema ser t-diagnosticável: duas unidades não se testam entre si; cada unidade é testada por pelo menos outras t unidades; e $N \geq 2t + 1$.

Outro modelo inicial para diagnóstico em nível de sistema é o modelo BGM, também nomeado pelas iniciais dos autores: Barsi, Grandoni e Maestrini [15]. Este modelo é similar ao modelo PMC, emprega o mesmo grafo de testes, mas assume diferentes resultados para os testes. As asserções básicas são: cada teste é executado por uma única

unidade; cada unidade é capaz de testar qualquer outra unidade; nenhuma unidade testa a si mesma. Considerando um teste da unidade u_i sobre a unidade u_j, o modelo de diagnóstico é definido como segue:

— se u_i é sem-falha, o resultado do teste é 0 se u_j for sem-falha; o resultado do teste é 1 se u_j for uma unidade falha;

— se u_i é falha e u_j é sem-falha, qualquer um dos dois resultados para o teste é possível; e

— se u_i e u_j são falhas, a saída do teste é necessariamente 1.

Neste modelo se o resultado do teste $a_{i,j} = 0$, isto é, a unidade i testa a unidade j como sem-falha, então é possível concluir que u_j *não* é falha; enquanto que se $a_{i,j} = 1$, não é possível que ambas as unidades u_i e u_j sejam unidades sem-falha. Nenhuma outra possibilidade pode ser excluída com base no resultado deste teste executado pelas unidade i sobre a unidade j. A Tabela 2.2 mostra as regras de invalidação para o modelo BGM.

Unidade Testadora	**Unidade Testada**	**Resultado**
sem-falha	sem-falha	0 (*pass*)
sem-falha	falha	1 (*fail*)
falha	sem-falha	0 ou 1
falha	falha	1 (*fail*)

Tabela 2.2: Regras de invalidação para o modelo BGM.

Em [15] os autores também apresentam as condições necessárias e suficientes para a t-diagnosticabilidade para ambos os sistemas diagnosticáveis em um passo, e os sistemas sequencialmente diagnosticáveis. Se cada unidade é testada por pelo menos outras t unidades, os autores mostram que a diagnosticabilidade em um passo é no máximo $N - 2$, ou seja, $N \geq t + 2$. No diagnóstico sequencial, que também é chamado de diagnóstico com reparação (ou *diagnosis with repair*), se uma unidade falha é encontrada, ela é reparada e o processo é então sequencialmente repetido até que todas as unidades falhas sejam diagnosticadas e reparadas. Em [160] os autores mostram que o diagnóstico sequencial para topologias arbitrárias de redes é co-NP-Completo. Posteriormente em [4] a diagnosticabilidade de grafos simétricos sobre o modelo BGM também foi determinada.

Um resultado importante sobre o diagnóstico em nível de sistema foi a introdução do diagnóstico *adaptativo* [151]. Os modelos anteriores consistiam em inicialmente escolher o conjunto de testes para ser executado, em executar aqueles testes pré-definidos, e finalmente em avaliar o resultado dos testes com o objetivo de identificar as unidades falhas. No diagnóstico adaptativo, o conjunto de testes a ser executado é determinado de forma dinâmica, baseado em resultados de testes anteriores.

O primeiro modelo de diagnóstico adaptativo foi apresentado pelos autores de [151] que também apresentam um algoritmo para este modelo. Assumindo um sistema S de N unidades onde não existem mais de t unidades falhas, o algoritmo proposto escolhe e executa de forma adaptativa os testes, repetindo o processo até que uma unidade sem-falha seja identificada. Então esta unidade é utilizada como uma unidade testadora a partir da qual todas as unidades falhas são então identificadas. Foi provado que $(N - 1) + t(t + 1)$ testes são suficientes para que todas as unidades falhas do sistema sejam identificadas. Na sequência, em [96], os autores apresentam outro algoritmo adaptativo onde o diagnóstico é realizado com no máximo $(N + 2t - 2)$ testes. Ambos os algoritmos adaptativos apresentados são completos e corretos. Um algoritmo é chamado de *completo* quando ele, considerando o limite máximo de unidades falhas permitidas t, consegue identificar todas as unidades falhas do sistema. Um algoritmo é chamado de *correto* quando prova-se que o estado – falho ou sem-falha – das unidades diagnosticadas pelo algoritmo é identificado corretamente.

No diagnóstico adaptativo e em todos os outros modelos anteriores, os resultados dos testes são coletados e processados por uma entidade externa, que também determina o estado de todas as unidades do sistema. No diagnóstico *distribuído* em nível de sistema, proposto por Kuhl e Reddy [120, 121, 119], as próprias unidades sem-falha do sistema diagnosticam o estado de todas as unidades. Estas unidades executam testes e trocam entre si informações sobre os resultados de testes. Os autores propuseram o algoritmo *SELF* de diagnóstico distribuído, ou seja, o algoritmo permite que todas as unidades sem-falha diagnostiquem, de forma independente, o estado de todas as unidades, desde

que o número total de falhas não exceda um dado limite t. Por outro lado, apesar de ser completamente distribuído, este algoritmo não é adaptativo, isto é, cada unidade tem uma configuração de testes fixa e previamente definida. Posteriormente, Hosseini, Kuhl e Reddy [103] estenderam o algoritmo *SELF*, introduzindo o algoritmo *NEW-SELF*, também um algoritmo distribuído mas com a diferença de que ele permite a reentrada no sistema de unidades falhas que foram reparadas ou substituídas. Esta nova versão também permite a adição de novas unidades ao sistema. O algoritmo *EVENT-SELF* também é um algoritmo distribuído e foi proposto por Bianchini, Goodwin e Nydick em [21]. Este algoritmo é uma extensão do algoritmo *NEW-SELF* e usa técnicas baseadas em eventos para reduzir a quantidade de recursos de rede necessários para a realização do diagnóstico.

O diagnóstico *adaptativo e distribuído* em nível de sistema foi proposto por Bianchini e Buskens [19, 20]. Os autores propõem o algoritmo *Adaptive-DSD* que é ao mesmo tempo distribuído e adaptativo. O *Adaptive-DSD* é executado em cada nodo do sistema em *intervalos de testes* pré-definidos. Cada nodo executa pelo menos um teste por intervalo de testes. Uma *rodada de testes* é definida como o período de tempo no qual todos os nodos do sistema executam todos os seus testes pelo menos uma vez. Todo nodo sem-falha alcança um diagnóstico consistente em no máximo N rodadas de testes. Até $N-1$ nodos podem estar falhos para que os nodos sem-falha continuem capazes de realizar o diagnóstico do sistema.

A cada intervalo de testes, um nodo sem-falha realiza testes sequencialmente até que outro nodo sem-falha seja encontrado, ou então até que o testador teste todos os nodos do sistema como falhos. O grafo de testes neste caso é um anel que conecta todos os nodos sem-falha, como mostrado na Figura 2.2. No exemplo mostrado nesta figura, os nodos 1, 4 e 5 estão falhos, e todos os demais nodos estão sem-falha. Como o nodo 0 testa o nodo 1 como falho, então, na sequência, o nodo 0 testa também o nodo 2. Como este último teste identifica que o nodo 2 está sem-falha, então o nodo 0 para e não realiza novos testes naquele intervalo de testes. Já o nodo 2 realiza apenas um teste – sobre o nodo 3 que está

sem-falha – e assim por diante.

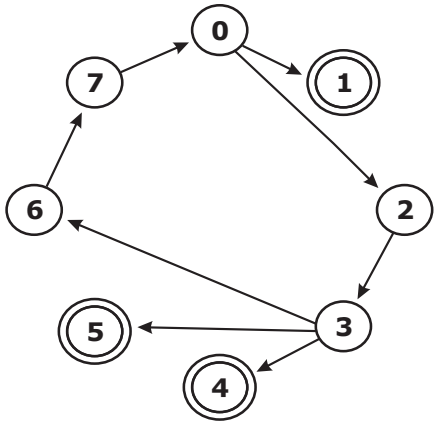


Figura 2.2: Exemplo de testes executados por nodos sem-falha no *Adaptive-DSD*.

Quando um testador executa um teste com sucesso, isto é, o nodo testado é sem-falha, o testador obtém *informações de diagnóstico* [56] a partir do nodo testado. Informações de diagnóstico são informações recebidas pelo nodo testador a partir do nodo testado, e incluem o estado de outros nodos do sistema. Define-se como *latência de diagnóstico* o número de rodadas de testes necessárias para que todos os nodos sem-falha completem o diagnóstico do sistema. O pior caso para a latência de diagnóstico do *Adaptive-DSD* é de N rodadas de testes. O *Adaptive-DSD* foi implementado e resultados práticos foram apresentados mostrando que o algoritmo é efetivo quando usado para monitorar redes locais.

Já o diagnóstico *hierárquico* em nível de sistema foi proposto com o objetivo de reduzir a latência do diagnóstico distribuído e adaptativo [55, 56, 54], ou seja, diminuir o número de rodadas de testes necessárias para que todos os nodos sem-falha completem o diagnóstico do sistema. No diagnóstico hierárquico, os nodos são agrupados em *clusters* virtuais de tamanho progressivo, e quando um nodo sem-falha é testado, o testador obtém informação de diagnóstico sobre todos os nodos do *cluster* ao qual o nodo testado pertence. Em [56] o algoritmo hierárquico, adaptativo e distribuído *Hi-ADSD* é apresentado – este algoritmo possui latência de diagnóstico de no máximo $log_2^2 N$ rodadas de testes para sistemas compostos de N nodos. Outro algoritmo de diagnóstico hierárquico e

adaptativo, o *Hi-ADSD with Timestamps* [54], constrói *clusters* sempre de tamanho $N/2$, estratégia que resulta em uma latência média menor do que a apresentada pelo algoritmo *Hi-ADSD*. É importante ressaltar que todos estes modelos e algoritmos de diagnóstico em nível de sistema apresentados acima, assim como no modelo PMC, assumem um sistema completamente conectado.

Apesar destes diversos modelos e algoritmos assumirem uma rede subjacente completamente conectada, alguns trabalhos foram propostos para topologias arbitrárias. Em [13] Bagchi e Hakimi introduzem um algoritmo distribuído de diagnóstico em nível de sistema para redes de topologia arbitrária. Por outro lado, o algoritmo não pode ser utilizado para monitoramento contínuo da rede pois a sua execução é feita de forma *off-line*. Em [173] Stahl, Buskens e Bianchini introduzem e avaliam, através de simulações, um novo algoritmo adaptativo e distribuído para sistemas de topologia arbitrária. Diferente de [13] o algoritmo proposto pode ser executado de forma *on-line*.

Rangarajan, Dahbura e Ziegler apresentam em [162] o algoritmo *RDZ* , um algoritmo distribuído para sistemas de topologia arbitrária. O algoritmo cria uma grafo de testes que possui número de testes ótimo, isto é, cada nodo possui um testador. Em [57] os autores também apresentam um algoritmo distribuído para redes de topologia arbitrária. O algoritmo é o primeiro que permite a cada nodo, além de determinar o estado de outras unidades, identificar quais partes da rede estão inalcançáveis. Em [176], Subbiah e Blough definem um *framework* teórico chamado *bounded correctness* no qual é possível provar a correção do diagnóstico distribuído na presença de falhas e reparações dinâmicas. Os autores apresentam um algoritmo distribuído para sistemas completamente conectados e outro para redes de topologias arbitrárias e provam seus respectivos limites de correção.

2.2 Diagnóstico Baseado em Comparações: Os Primeiros Modelos

O primeiro modelo de diagnóstico baseado em comparações foi proposto por Malek [143]. Este modelo é baseado na comparação das saídas produzidas pela execução de tarefas por pares de unidades. A comparação que resulta em diferença indica que uma ou ambas as unidades comparadas estão falhas. Note que é possível que ambas as unidades estejam falhas, e neste caso, a comparação deve indicar diferença. Este modelo assume que:

1. As saídas produzidas por duas unidades sem-falha em resposta a uma mesma tarefa são sempre idênticas;
2. A saída produzida por uma unidade falha é sempre diferente das saídas produzidas por qualquer outra unidade (falha ou sem-falha) para a mesma tarefa.

Este modelo consiste em duas atividades: detecção de falhas e localização de falhas. O objetivo da detecção de falhas é somente determinar a presença de unidades falhas no sistema, mas não determina quais são as unidades falhas. Já a localização de falhas permite a identificação das unidades que estão falhas.

Um sistema S com N unidades é modelado como um grafo $G = (V, E)$ que é um grafo conectado, isto é, existe um caminho entre todo par de vértices. Neste grafo, V é um conjunto de N vértices e E é um conjunto de arestas. Cada vértice do conjunto V corresponde a um processador ou uma unidade do sistema. Cada aresta em E representa a conexão ou o enlace (*link*) de comunicação entre um par de unidades.

Este modelo assume que as tarefas são executadas por pares de unidades diferentes. O modelo também assume a existência de um *observador central* que coleta e mantém informações sobre as saídas das tarefas. Este observador central realiza o diagnóstico do sistema baseado nos resultados das comparações, detectando a existência de falhas e determinando quais são as unidades falhas do sistema. O observador central é uma

unidade confiável e que nunca falha. Quando as saídas de duas unidades são comparadas, os possíveis resultados para a comparação são mostrados na Tabela 2.3. O conjunto dos possíveis resultados de comparações é também chamado de regras de invalidação (*invalidation rules*). Se a comparação resultar em *igualdade* ambas as unidades estão sem-falha, enquanto que se a comparação resultar em *diferença* ao menos uma das duas unidades está falha. Neste caso, mais comparações são necessárias para identificar a unidade falha.

Unidade 1	**Unidade 2**	**Resultado da Comparação**
sem-falha	sem-falha	0 (igualdade)
sem-falha	falha	1 (diferença)
falha	sem-falha	1 (diferença)
falha	falha	1 (diferença)

Tabela 2.3: Regras de invalidação para o modelo apresentado por Malek.

Em [143] também é provado que, em um sistema com N unidades no qual é possível comparar qualquer par de unidades, o número máximo de unidades falhas é $N-2$ para que o diagnóstico seja correto, isto é, a diagnosticabilidade (*diagnosability*) é $N-2$. Como um exemplo, a Figura 2.3 mostra um grafo G com 4 vértices e 6 arestas. Considerando a unidade 1 como falha, a Tabela 2.4 mostra o resultado para todas as possíveis comparações neste sistema. Nesta tabela pode-se notar que todas as comparações que envolvem a unidade 1 resultam em diferença (primeira, quarta e quinta linhas da tabela), enquanto que as demais comparações que envolvem apenas unidades sem-falha resultam todas em igualdade.

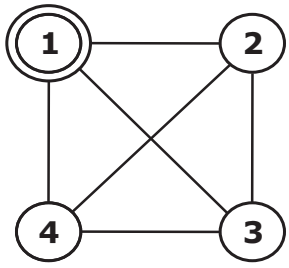


Figura 2.3: Um grafo exemplo representando um sistema com 4 unidades; a unidade 1 é uma unidade falha.

Id. da Unidade	Id. da Unidade	Resultado da Comparação
1	2	1 (diferença)
2	3	0 (igualdade)
3	4	0 (igualdade)
1	3	1 (diferença)
1	4	1 (diferença)
2	4	0 (igualdade)

Tabela 2.4: Um exemplo de todos os resultados das comparações para um sistema de 4 unidades quando a unidade 1 é falha.

Chwa e Hakimi em [42] propuseram outro modelo de diagnóstico baseado em comparações, similar ao proposto por Malek. Neste modelo o sistema de N unidades também é representado por um grafo $G = (V, E)$. Uma mesma tarefa também é enviada para as unidades em pares. O estado das unidades – falha ou sem-falha – é determinado através da comparação do resultado das tarefas. Se a comparação resultar em diferença, pelo menos uma unidade é falha, como mostrado na Tabela 2.5. Este modelo também assume um observador central que realiza o diagnóstico completo do sistema baseado no resultado das comparações.

A diferença entre esse modelo e o modelo anterior é que, nesse modelo, quando duas unidades falhas recebem a mesma tarefa para executar, elas podem produzir a mesma saída, ou seja, nesse modelo a comparação das saídas de duas tarefas produzidas por duas unidades falhas pode resultar em igualdade.

Unidade 1	Unidade 2	Resultado da Comparação
sem-falha	sem-falha	0 (igualdade)
sem-falha	falha	1 (diferença)
falha	sem-falha	1 (diferença)
falha	falha	0 ou 1

Tabela 2.5: Regras de invalidação para o modelo apresentado por Chwa e Hakimi.

Barborak, Malek e Dahbura em [14] apresentam um *survey* dos primeiros modelos de diagnóstico baseado em comparações. Neste trabalho o diagnóstico é tratado como um *framework* unificado junto com outros problemas e algoritmos distribuídos, incluindo os problemas de consenso e dos Generais Bizantinos. Entre as contribuições de [14], uma

classificação de tipos de falhas é apresentada, incluindo a especificação da classe de falhas por computação incorreta (*incorrect computation fault*) – que é a melhor descrição das falhas que podem ser tratadas pelo diagnóstico baseado em comparações. Esta descrição é importante porque vários artigos anteriores de diagnóstico apenas apresentam implicitamente a classe de falhas assumida, especificando apenas como as falhas são detectadas. O artigo também argumenta que, se a frequência com que duas unidades ficam falhas é baixa, então existe uma pequena probabilidade de que ambas as unidades se tornem falhas no mesmo momento. Assim, duas unidades executando a mesma tarefa devem produzir resultados idênticos a menos que uma, ou ambas as unidades, se tornem falhas.

Na sequência, esta seção tem duas subdivisões e realiza, nesta ordem, uma breve apresentação das classes de falhas descritas por Barborak, Malek e Dahbura, e uma descrição de extensões e avaliações apresentadas sobre os primeiros modelos de diagnóstico baseados em comparações de Malek e de Chwa e Hakimi.

2.2.1 Classes de Falhas

Esta subseção faz uma breve descrição das classes de falhas apresentadas pelos autores de [14, 126]. Em outras palavras, os autores apresentam uma classificação das razões pelas quais uma unidade pode se tornar falha. As classes são apresentadas de uma forma na qual uma classe menos abrangente é um subconjunto de outra classe mais abrangente.

A Figura 2.4 faz uma apresentação ordenada destas classes de falhas. As classes apresentadas – da menos abrangente para a mais abrangente – são: falhas *fail-stop*, falhas *crash*, falhas de omissão, falhas *timing*, falhas de computação incorreta, e falhas Bizantinas, e são descritas da seguinte forma:

— Uma falha *fail-stop* ocorre quando uma unidade deixa de funcionar e outras unidades determinam esta situação de falha.

— Uma falha *crash* ocorre quando uma unidade para de funcionar e perde seu estado interno.

— Uma falha de omissão (*omission*) ocorre quando uma unidade não consegue iniciar ou finalizar uma tarefa.

— Uma falha *timing* ocorre quando uma unidade completa uma tarefa em um intervalo de tempo diferente do especificado.

— Uma falha de computação incorreta (*incorrect computation fault*) ocorre quando uma unidade não produz o resultado correto em resposta à uma tarefa correta recebida como entrada.

— Uma falha bizantina (*Byzantine*) é uma falha arbitrária, por exemplo maliciosa; uma falha Bizantina também pode ser descrita como qualquer tipo possível de falha.

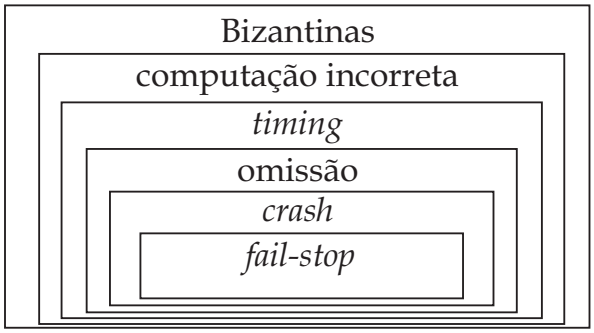


Figura 2.4: Uma apresentação ordenada das classes falhas introduzidas em [14, 126].

2.2.2 Primeiros Modelos: Extensões e Avaliações

Ammann e Dal Cin em [8] também investigam a diagnosticabilidade do diagnóstico baseado em comparações, mostrando que a condição necessária para um sistema ser t-diagnosticável é que cada nodo do grafo de testes tenha grau no mínimo t; um grau mínimo estritamente maior que t é uma condição suficiente. O grau – ou ordem – de um nodo é o número de arestas adjacentes a este nodo. Ammann e Dal Cin também apresentaram um algoritmo para o diagnóstico sequencial para um subconjunto dos sistemas t-diagnosticáveis. A complexidade do algoritmo proposto é $O(N^2)$. Eles também

introduziram um algoritmo paralelo para o diagnóstico quando a topologia é uma árvore [47, 8].

Yang e Masson em [194] apresentaram um modelo de comparações considerando o diagnóstico de falhas para multiprocessadores aplicado para sistemas t_1/t_1-diagnosticáveis. O sistema é dito ser t/s-diagnosticável se, na presença de no máximo t falhas, todas as unidades falhas podem ser identificadas através da substituição de no máximo s unidades [84]. Os sistemas t_1/t_1-diagnosticáveis são um caso especial dos sistemas t/s-diagnosticáveis, onde $s = t_1$ [41]. Em um sistema t_1/t_1-diagnosticável todas as unidades falhas, exceto no máximo uma, são corretamente identificadas. Em outras palavras, no máximo uma unidade falha é incorretamente diagnosticada como sendo sem-falha. Assim como no modelo de Chwa e Hakimi, o modelo proposto por Yang e Masson assume que a comparação de duas unidades falhas pode resultar em igualdade. Eles também apresentam um algoritmo $O(|C|)$ para o modelo de comparações para sistemas t_1/t_1-diagnosticáveis, onde C representa o conjunto de todas as comparações realizadas.

Xu e Huang [192] caracterizaram a $t/(N-1)$-diagnosticabilidade para vários tipos de estruturas sobre o modelo de Chwa e Hakimi. O sistema com N unidades é $t/(N-1)$-diagnosticável se no máximo t unidades são falhas e se as unidades falhas estiverem em um conjunto de tamanho $(N-1)$. Os autores apresentam uma síntese das configurações ótimas $t/(N-1)$-diagnosticáveis para várias topologias, como cadeias (*chains*) e *loops*. Em particular foi mostrado que para $N = 2t+1$, as cadeias são $t/(N-1)$-diagnosticáveis se $N \geq 9$ e os *loops* são diagnosticáveis para $N \geq 13$. Posteriormente Xu e Randell [193] aplicaram o diagnóstico $t/(N-1)$ para projetos de software. Eles propuseram uma técnica de programação $t/(N-1)$-variante que diagnostica falhas em *frameworks* de software redundantes.

Kozlowski e Krawczyk [117] estenderam o modelo de diagnóstico de Chwa e Hakimi para situações com falhas híbridas. Uma situação com falha híbrida é definida como t/m-restrita se o número de unidades falhas não exceder t e o número de resultados errôneos de comparações for menor que m. Um resultado errôneo de comparação ocorre quando

uma unidade falha acaba erroneamente sendo identificada como sem-falha – os autores citam que esta situação pode ocorrer, por exemplo, quando um resultado reportado está incompleto. Kozlowski e Krawczyk também apresentam um algoritmo $O(N\,|C|)$ para o diagnóstico baseado em comparações sobre situações com falhas híbridas, onde C representa o conjunto de todas as comparações realizadas em sistemas com N unidades.

Fuhrman e Nussbaumer em [85, 86] apresentam o modelo *Bounded Symmetric Comparison* (BSC) para o diagnóstico em nível de sistema baseado em comparações. Este modelo é baseado no modelo de Chwa e Hakimi [42] mas inclui um limite no número de nodos falhos que podem produzir resultados idênticos. No modelo BSC f_1 representa o número máximo de nodos que podem ser falhos, e f_2 é o limite superior do número de nodos falhos que podem produzir resultados idênticos. Além disso, $f_2 \leq f_1$. Os autores provam as condições necessárias e suficientes para o diagnóstico em um passo (*one-step*) para um sistema sobre o modelo BSC. Eles mostram que um sistema é diagnosticável em um passo se e somente se para todo par distinto de conjuntos F_1, F_2 onde $F_1 \subset V, F_2 \subset V$ e $|F_1| \leq f_1, |F_2| \leq f_1$, uma das seguintes condições é satisfeita:

— Existe uma aresta entre um nodo em $V - (F_1 \cup F_2)$ e um nodo em $(F_1 \cup F_2) - (F_1 \cap F_2)$.

— Um componente do grafo corresponde a $F_1 - (F_1 \cap F_2)$, ou então a $F_2 - (F_1 \cap F_2)$.

Kreutzer e Hakimi [118, 136] apresentam dois modelos de diagnóstico baseado em comparações – chamados KH1 e KH2 – que são baseados respectivamente nos modelos de Chwa e Hakimi e de Malek. Apesar de serem baseados nos modelos de Chwa e Hakimi e de Malek, os modelos KH1 e KH2 consideram a possibilidade do comparador das unidades testadas estar falho, e ainda avaliam de forma separada as falhas de comparadores das falhas das unidades testadas. No primeiro modelo (KH1), a comparação das saídas das tarefas produzidas por duas unidades falhas pode resultar em igualdade, e no segundo modelo (KH2) se o resultado da comparação das saídas resultar em igualdade, ambas as unidades

são consideradas sem-falha. Pelc em [157] argumenta que estes modelos são de fato equivalentes aos modelos propostos respectivamente por Chwa e Hakimi, e por Malek. Kreutzer e Hakimi também apresentam a caracterização para um sistema ser $(t-t_c)$-diagnosticável sobre estes dois modelos KH1 e KH2, onde um sistema $(t-t_c)$-diagnosticável é um sistema com no máximo t unidades falhas e no máximo t_c comparadores falhos. Eles mostram que um sistema S é $(t-t_c)$-diagnosticável se e somente se S é t-diagnosticável e $t_c < |\Gamma(i)|/2$, onde $\Gamma(i) \leq \Gamma(j) \mid \forall j \in V$ e $\Gamma(i) = \{j \mid i$ e j são comparadas$\}$, ou seja, $\Gamma(i)$ representa o conjunto das unidades comparadas com i, e consequentemente t_c deve ser um número menor que a metade da menor quantidade de comparações realizadas por qualquer unidade do sistema.

Pelc em [157] realiza uma análise de ambos os modelos baseado em comparações de Malek e de Chwa e Hakimi, que o autor denomina respectivamente de modelo assimétrico e modelo simétrico. Nesta análise, Pelc apresenta o pior caso do número de testes de algoritmos ótimos – considerando ambos os modelos – para o diagnóstico de t falhas, para o diagnóstico sequencial de t falhas, e para o diagnóstico em um passo de t falhas. O autor também considera testes adaptativos e não adaptativos e mostra que usando testes adaptativos o número de testes é frequentemente menor. Estes resultados são apresentados na sequência.

O número mínimo de testes para se completar o diagnóstico de t unidades falhas, onde $t \leq N$, sobre o modelo de Malek é $\lceil N/2 \rceil$. No caso do diagnóstico sequencial para t falhas (que identifica pelo menos uma unidade falha), onde $t \leq N-2$, o número mínimo de testes requeridos é $\max(\lfloor N/2 \rfloor * t) + 1$ quando a estratégia adaptativa para os testes é empregada, e $N - \lfloor N/(t+2) \rfloor$ para o diagnóstico não adaptativo. No caso do diagnóstico adaptativo em um passo para t falhas (que identifica todas as unidades falhas em um passo), para $t \leq N-2$ o número mínimo de testes é $O(N^2/(N-t))$ e, quando $N \geq 2t+1$ o número mínimo de testes é $\lfloor N/2 \rfloor + 3,5\lceil t/2 \rceil + 3$. Para o diagnóstico não adaptativo em um passo onde $t \leq N-2$, o número mínimo de testes é $O(Nt)$.

O número mínimo de testes para completar o diagnóstico de t unidades falhas, onde $t \leq$

$N-1$, no modelo de Chwa e Hakimi é $N-\lfloor N/(t+1)\rfloor$. No caso do diagnóstico sequencial de t falhas, onde $t < N/2$, o número mínimo de testes requeridos é $N - \lceil N/(t+1)\rceil + 1$ quando a estratégia adaptativa para os testes é empregada, e $N - \lfloor N/(2t+1)\rfloor$ para o diagnóstico não adaptativo. No caso do diagnóstico adaptativo em um passo para o diagnóstico de t falhas, onde $t < N/2$, o número mínimo de testes é $O(N)$ [118]. Para o diagnóstico não adaptativo de t falhas, se $t < N/2$, o número de testes é $O(Nt)$.

2.3 Modelo MM de Diagnóstico Baseado em Comparações

O modelo MM de diagnóstico baseado em comparações foi proposto por Maeng e Malek [140] para sistemas compostos de múltiplos processadores. O modelo considera que um sistema S de N unidades é representado por um grafo $G = (V, E)$, onde V representa o conjunto de unidades e E representa o conjunto de enlaces de comunicação. Cada aresta $(i, j) \in E$ indica que a unidade i está conectada (ou é vizinha) da unidade j, e vice versa. A Figura 2.5 mostra um exemplo de grafo para um sistema de 4 unidades. No modelo MM, o estado das unidades – falho ou sem-falha – é determinado através da comparação da saída de uma tarefa executada por uma unidade com a saída da mesma tarefa executada por outra unidade. O conjunto de todas as unidades falhas é representado por F. Para uma unidade $i \in V$, $N(i) = \{j \mid (i, j) \in E\}$ denota o conjunto de unidades vizinhas de i e $d(i) = |N(i)|$ é a ordem – ou o grau – da unidade i, isto é, ambos $d(i)$ e $|N(i)|$ correspondem ao número de vizinhos de i.

A principal diferença do modelo MM para os modelos anteriores [143, 42] é que ele permite que a comparação das saídas das tarefas seja realizada pelas próprias unidades do sistema, isto é, as unidades são também comparadoras. Uma unidade i é uma unidade comparadora das unidades j e k somente se $(i, j) \in E$ e $(i, k) \in E$; além disso $i \neq j$ e $i \neq k$. Os resultados das comparações são também enviados a um observador central que realiza o diagnóstico completo do sistema. O conjunto com todos os resultados das comparações é chamado de *síndrome* do sistema – ou *síndrome de comparações* – e é

representado por σ.

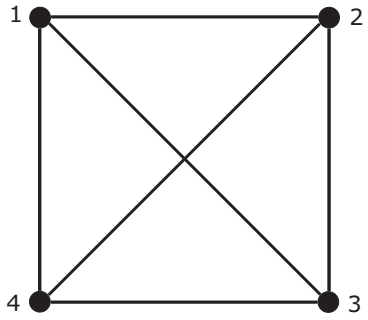


Figura 2.5: Exemplo de um grafo representando um sistema de 4 unidades.

O modelo MM representa os testes – ou comparações – realizados no sistema através de um multigrafo $M = (V, C)$ definido sobre o mesmo conjunto de unidades do grafo G. Cada aresta $(j, k)_i \in C$ representa a comparação das saídas para tarefas enviadas às unidades j e k pela unidade i; C representa o conjunto de todas as comparações realizadas. M é um multigrafo porque as saídas de um par de unidades podem ser comparadas por mais de uma unidade do sistema, ou seja, mais de uma aresta pode existir entre um mesmo par de vértices. A Figura 2.6 mostra um exemplo de multigrafo para um sistema de 4 unidades. Na figura pode-se notar que as unidades 3 e 4 são comparadas pela unidade 1 e também pela unidade 2, portanto, duas das comparações realizadas no sistema são as comparações $(3, 4)_1$ e $(3, 4)_2$.

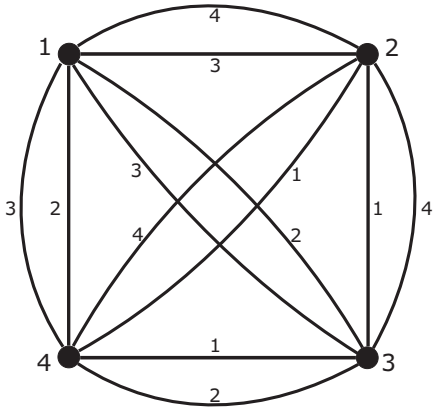


Figura 2.6: Um multigrafo M para um sistema de 4 unidades.

A notação $r((j, k)_i)$ é usada para representar o resultado da comparação das saídas das unidades j e k pela unidade i. O resultado é 0 quando a comparação indica igualdade

e o resultado é 1 quando a comparação indica diferença. Se $r((j,k)_i) = 1$, pelo menos uma das unidades i, j ou k está falha. Se $r((j,k)_i) = 0$ e a unidade testadora i está sem-falha então j e k estão também sem-falha. Mas se a unidade testadora i está falha, o resultado da comparação não é confiável e não se pode obter nenhuma conclusão sobre o estado das unidades j e k. Todos os possíveis resultados de comparações são apresentados na Tabela 2.6.

As principais asserções do modelo MM são:

— Toda falha é permanente, isto é, as unidades não se recuperam das falhas.

— A comparação realizada por qualquer unidade falha não é confiável.

— Duas unidades falhas que executam a mesma tarefa sempre retornam saídas diferentes.

— Toda unidade falha sempre gera resultados incorretos para cada tarefa de entrada, isto é, a comparação das saídas de tarefas produzidas por uma unidade falha e qualquer outra unidade (falha ou sem-falha) sempre resulta em diferença.

— Existe um limite t, que é o número máximo de unidades que podem estar falhas para que o diagnóstico do sistema seja possível.

Comparador	**Unidade 1**	**Unidade 2**	**Resultado da Comparação**
sem-falha	sem-falha	sem-falha	0 (igualdade)
sem-falha	sem-falha	falha	1 (diferença)
sem-falha	falha	sem-falha	1 (diferença)
sem-falha	falha	falha	1 (diferença)
falha	sem-falha	sem-falha	0 ou 1
falha	sem-falha	falha	0 ou 1
falha	falha	sem-falha	0 ou 1
falha	falha	falha	0 ou 1

Tabela 2.6: Regras de invalidação para o modelo MM.

A Figura 2.7 mostra um exemplo dos resultados de comparações para o multigrafo M da Figura 2.6. Cada aresta possui dois rótulos (*labels*), um representa a unidade que

compara o resultado das tarefas produzidas pelas unidades conectadas a cada aresta, e o segundo – mostrado dentro dos círculos – representa o resultado de cada comparação. Como exemplo, as unidades 3 e 4 são comparadas pela unidade 1 e o resultado da comparação indica igualdade – o valor 0 dentro do círculo.

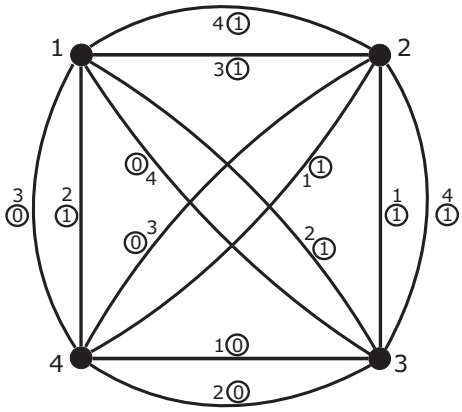


Figura 2.7: Multigrafo M mostrando o resultado das comparações para um sistema de exemplo com 4 unidades.

Além de apresentar o modelo de diagnóstico baseado em comparações, [140] também mostra que, para diagnosticar corretamente um sistema com 1 nodo falho ($t = 1$), o número total de unidades no sistema (N) deve ser maior que 3. Para $t \geq 2$, N deve ser maior ou igual a $2t + 1$. As condições necessárias e suficientes para diagnosticar até t nodos falhos neste modelo são: (1) o grau de todo nodo deve ser no mínimo t; (2) para todo par de nodos i, j que possuem distância igual a 1 ou 2, no máximo t nodos de um conjunto $W_{i,j}$ devem ser removidos para que o par de nodos e os seus vizinhos sejam desconectados do restante do grafo, e, além disso (3) não existe $W^*_{i,j}$ se $W_{i,j}$ possui exatos t nodos. $W^*_{i,j}$ é definido como um $W_{i,j}$ que possui no mínimo um par de vértices r, s tal que $W_{i,r} = (W_{i,j} - r) \cup j$ e $W_{j,s} = (W_{i,j} - s) \cup i$.

Maeng e Malek em [140] também apresentam o procedimento abaixo que constrói o grafo mínimo para diagnosticar um sistema $S_{t,N}$, com $t \geq 4$ e $N = 2t + 1$. Para $t = 1$ ou 2, o grafo mínimo é o grafo completo. Para $t = 3$ os autores mostram que o número de arestas deve ser pelo menos 14.

1. Se t é par, então seja $t = 2r$. $S_{2r,N}$ possui arestas conectando vértices i, j

tal que $i - r \leq j \leq i + r$, módulo N. Os nodos possuem identificadores sequenciais iniciando em zero.

2. Se t é ímpar e N é par, então seja $t = 2r + 1$. $S_{2r+1,N}$ possui uma aresta conectando o vértice i ao vértice $i + (N/2)$ em $S_{2r,N}$, $1 \leq i \leq N/2$.

3. Se t é ímpar e N também é ímpar, então seja $t = 2r + 1$. $S_{2r+1,N}$ possui todas as arestas em $S_{2r,N}$ mais as arestas do vértice 0 ao vértice $(N-1)/2$ e ao vértice $(N + 1)/2$, e também do vértice i ao vértice $i + (N + 1)/2$ para $1 \leq i < (N - 1)/2$.

A Figura 2.8 mostra o $S_{4,9}$.

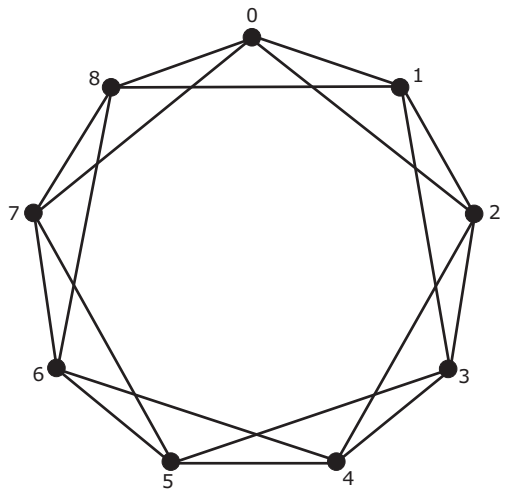


Figura 2.8: O grafo mínimo $S_{4,9}$.

Uma discussão sobre a latência de diagnóstico no modelo MM também é apresentada – vale lembrar que a *latência de diagnóstico* é o número de ciclos de testes necessários para que todas as unidades sem-falha completem o diagnóstico do sistema. Primeiramente assume-se que cada comparador pode executar somente uma comparação por unidade de tempo. Um *ciclo de testes* é definido como uma aplicação do número máximo de comparações no sistema, que consiste de $\lfloor N/3 \rfloor$ comparações simultâneas, já que cada unidade é comparadora ou é comparada. Os autores mostram que o limite inferior para o número mínimo de ciclos de testes é $\lceil \lceil Nt/2 \rceil / \lfloor N/3 \rfloor \rceil$, onde $\lceil Nt/2 \rceil$ é o número mínimo de comparações quando cada vértice possui grau t.

2.3.1 O Modelo MM*

Maeng e Malek também em [140] apresentam um caso especial do modelo MM, chamado de *modelo MM**, ou seja, o modelo MM abrange o modelo MM*. A diferença é que no modelo MM* cada unidade compara *todo par* de unidades vizinhas a que estão conectadas. Como exemplo, a Figura 2.9 mostra um sistema onde o testador, a unidade 2, no modelo MM*, realiza todas as seguintes comparações: $(1,3)_2$, $(1,5)_2$ e $(3,5)_2$. Os resultados das comparações são então enviados pela unidade testadora ao observador central que completa o diagnóstico.

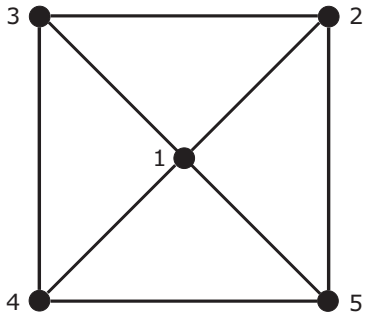


Figura 2.9: Um exemplo de um sistema com 5 unidades.

Sengupta e Dahbura em [169] generalizaram o modelo MM permitindo que as unidades comparadoras sejam ao mesmo tempo uma das unidades comparadas. Eles também apresentam uma caracterização de sistemas diagnosticáveis para o modelo MM. É importante salientar que quando a unidade comparadora sempre compara si mesma com outra unidade, a configuração de comparações é equivalente à configuração de testes do modelo PMC onde um testador realiza um teste sobre uma outra unidade. Neste sentido este modelo generaliza o modelo PMC.

Além disso, os autores apresentam um algoritmo polinomial para identificar falhas de processadores em sistemas de topologia arbitrária no qual os processadores realizam comparações para todo par de vizinhos. Mais importante, eles mostram que a diagnosticabilidade de sistemas de topologia arbitrária sobre este modelo é um problema NP-completo.

A seguir, esta seção descreve as condições necessárias apresentadas por Sengupta e Dahbura [169] para um sistema ser t-diagnosticável sobre o modelo MM. Como estas

condições são importantes para o algoritmo proposto no Capítulo 3, além de descritas de forma detalhada a seguir, um resumo destas condições também é apresentado na Seção 3.1.

2.3.2 t-Diagnosticabilidade (t-*Diagnosability*)

Sengupta e Dahbura [169] solucionaram o problema da t-diagnosticabilidade (t-*Diagnosability*) para um certo valor inteiro t sobre o modelo MM. É importante lembrar que um sistema é t-diagnosticável se todas as unidades falhas do sistema puderem ser identificadas desde que o número de unidades falhas não seja maior que t. Além disso, em um sistema t-diagnosticável, para cada síndrome existe um conjunto único de unidades falhas que pode produzir aquela síndrome, desde que o número de unidades falhas não seja maior que t.

Sejam S_1 e S_2 conjuntos de unidades. Um par (S_1, S_2) tal que $S_1, S_2 \subset V$ e $|S_1|, |S_2| \leq t$ é definido como *distinguível* ou *indistinguível* da seguinte forma. Seja $\sigma(F)$ o conjunto de síndromes que podem ser geradas se F é o conjunto de nodos falhos. O par de conjuntos $S_1, S_2 \mid S_1 \neq S_2$ é dito ser indistinguível se e somente se $\sigma(S_1) \cap \sigma(S_2) \neq \emptyset$; caso contrário ele é considerado distinguível.

Para provar que um par (S_1, S_2) é distinguível, pelo menos uma das seguintes três condições devem ser satisfeitas:

1. $\exists i, j \in V - S_1 - S_2$ e $\exists k \in (S_1 - S_2) \cup (S_2 - S_1)$ tal que $(j, k)_i \in C$;
2. $\exists i \in V - S_1 - S_2$ e $\exists j, k \in S_1 - S_2$ tal que $(j, k)_i \in C$;
3. $\exists i \in V - S_1 - S_2$ e $\exists j, k \in S_2 - S_1$ tal que $(j, k)_i \in C$.

Sengupta e Dahbura provam que um sistema S com N nodos é t-diagnosticável se e somente se para cada par de conjuntos $S_1, S_2 \in V \mid S_1 \neq S_2$ e $|S_1|, |S_2| \leq t$, (S_1, S_2) é um par distinguível. Em outras palavras, considerando o conjunto $\sigma(S_1)$ que é o conjunto de síndromes que podem ser produzidas se S_1 é o conjunto de nodos falhos e considerando o conjunto $\sigma(S_2)$ analogamente definido, $\sigma(S_1) \cap \sigma(S_2) = \emptyset$.

Eles também provam que para um sistema com N nodos ser t-diagnosticável, $N \geq 2t+1$ e cada nodo deve possuir grau maior ou igual a t, isto é, a saída de cada nodo deve ser comparada com as saídas de pelo menos outros t nodos. Além disso, para cada conjunto $X \subset V$ tal que $|X| = N - 2t + p$ e $0 \leq p \leq t-1$, os autores provam que $|T(X)| > p$, onde $T(X) = \{k \mid (j,k)_i \in C \text{ e } i,j \in X \text{ e } k \in V - X\}$. Em outras palavras: para um dado conjunto $X \subset V$, $T(X)$ denota o conjunto de unidades em $V - X$ que são comparadas *com* alguma unidade em X e *por* algum unidade de X.

Sengupta e Dahbura também definem um conjunto $U \subset V$ como um AFS (*Allowable Fault Set*), ou *possível conjunto de unidades falhas*, para a síndrome σ de S, se para quaisquer três unidades i,j,k tal que $(j,k)_i \in C$, $j,k \in N(i)$ e $j \neq k$, as seguintes condições são satisfeitas:

— se $i \in V - U$ e $j,k \in V - U$ então $r((j,k)_i) = 0$

— se $i \in V - U$ e $\{j,k\} \cap U \neq \emptyset$ então $r((j,k)_i) = 1$

Em outras palavras, para verificar se um conjunto U é um AFS, para cada comparação $(j,k)_i$ realizada no sistema, a primeira condição indica que: se a unidade comparadora i não estiver no AFS e também ambas as unidades j,k comparadas também não estiverem no AFS – ou seja $i,j,k \notin U$ – então o resultado da comparação $(j,k)_i$ deve indicar igualdade. Além disso, com base na segunda condição, se unidade comparadora i não estiver no AFS mas ao menos uma das unidades comparadas j,k estiverem no AFS, então o resultado da comparação deve indicar diferença. Como exemplo, considere um sistema com 8 unidades, onde $U = \{u_6, u_7, u_8\}$ e as comparações $(u_2,u_3)_{u_1}$ e $(u_2,u_6)_{u_1}$ são duas das comparações realizadas no sistema. Se o resultado destas comparações forem respectivamente $r((u_2,u_3)_{u_1}) = 1$ e $r((u_2,u_6)_{u_1}) = 1$, então o conjunto $U = \{u_6,u_7,u_8\}$ não é um AFS, pois a primeira condição não é satisfeita. Por outro lado, quando o resultado destas comparações forem $r((u_2,u_3)_{u_1}) = 0$ e $r((u_2,u_6)_{u_1}) = 0$, então o mesmo conjunto U também não é um AFS, pois agora a segunda condição não é satisfeita.

Por fim, para a síndrome σ, um AFS de cardinalidade mínima é chamado de um *AFS*

mínimo de σ, e denotado por $MASF(\sigma)$. Denota-se por t-AFS um conjunto AFS com no máximo t unidades. Os autores então apresentam que dado um sistema t-diagnosticável com no máximo t unidades falhas e uma síndrome de comparações σ, resolver o problema do diagnóstico do sistema é encontrar um $MASF(\sigma)$. Além disso, como em um sistema t-diagnosticável o conjunto de unidades falhas F é único e $|F| \leq t$, então existe apenas um único conjunto AFS como no máximo t unidades, isto é, existe apenas um t-AFS.

2.3.2.1 t/x-Diagnosticabilidade e $t[x]$-Diagnosticabilidade

Sengupta e Rhee em [170] definem a t/x-diagnosticabilidade e a $t[x]$-diagnosticabilidade. Um sistema é t/x-diagnosticável se todos os processadores falhos podem ser identificados unicamente a partir do conjunto dos resultados das comparações sempre que não exista mais de t processadores falhos e que a quantidade de resultados de comparações ausentes seja no máximo x. Os autores consideram a t/x-diagnosticabilidade para casos onde o resultado de uma comparação pode estar ausente devido a uma falha na transmissão da tarefa de entrada ou ainda na transmissão do resultado da tarefa executada. Um sistema é $t[x]$-diagnosticável se todos os processadores falhos podem ser identificados unicamente a partir do conjunto dos resultados das comparações sempre que não exista mais de t processadores falhos e que a quantidade de resultados de comparações que levem a identificação incorreta seja no máximo x. Este último conceito é usado para representar, por exemplo, nodos com falhas intermitentes.

Sejam dois conjuntos de processadores $S_1, S_2 \cup V$, $X(S_1, S_2) = \{(i,j)_k \mid k \in S_1 \text{ e } \{i,j\} \subset S_1 \cup S_2 \text{ e } \{i,j\} \cap S_2 \neq \emptyset\}$. Em outras palavras, $X(S_1, S_2)$ denota o conjunto de comparações onde o comparador está em S_1 e um dos processadores comparados está em S_2 e o outro processador comparado está em $S_1 \cup S_2$. É provado que um sistema é t/x-diagnosticável se e somente se, para todo $S_1, S_2 \subset V$, tal que $|S_1|, |S_2| \leq t$, $CT(V - S_1 - S_2, S_1 - S_2) + CT(V - S_1 - S_2, S_2 - S_1) > x$ onde $CT(S_1, S_2)$ denota a cardinalidade do conjunto $X(S_1, S_2)$. Também é provado que um sistema é $t[x]$-diagnosticável se e somente se: (a) para todo $S_1 \subset V$, tal que $|S_1| = t$, e para todo

$i \in S_1$, $CT(V - S_1, \{i\}) > x$; e, (b) para todo S_1, S_2, tal que $S_1, S_2 \subset V$, e $|S_1| = |S_2| = t$, pelo menos uma das seguintes condições é satisfeita:

— $CT(V - S_1 - S_2, S_1 - S_2) > x$;

— $CT(V - S_1 - S_2, S_2 - S_1) > x$.

2.3.2.2 Outras Extensões do Modelo MM

Em [33] uma extensão do modelo MM é apresentada. Este modelo considera ambas as falhas de processadores que são comparados e de comparadores de forma separada. Um processador ou executa tarefas ou realiza comparações. Os autores mostram que a diagnosticabilidade do sistema é $t \leq \lfloor \delta/2 \rfloor$, onde δ é o grau do nodo de menor grau no sistema. Entretanto, eles também mostram que se o número de comparadores falhos é menor que a quantidade dos outros processadores falhos, a diagnosticabilidade é de $t \leq \delta$. Os autores também apresentam um algoritmo ótimo $O(|E^*|)$ para o diagnóstico se $t \leq \lfloor \delta/2 \rfloor$, e um algoritmo $O(|E^*|^2)$ para o diagnóstico se $t \leq \delta$, onde E^* é o conjunto de comparadores.

Em [166] falhas de unidades comparadoras e também do observador central são consideradas. Para realizar o diagnóstico das unidades comparadoras, os autores propõem uma estratégia para executar exaustivamente comparações entre unidades sem-falha e unidades comparadoras. Estes testes são realizados testando diferentes tarefas de entrada e assume-se que uma unidade, mesmo que esteja falha, sempre produz a mesma resposta para uma mesma tarefa de entrada. Os autores aplicam a abordagem proposta para circuitos integrados, apresentando uma solução de um projeto de circuitos com bom custo-benefício [166]. Em [141] os autores apresentam uma solução de diagnóstico, baseado no modelo MM, que é aplicada para a localização de falhas em *arrays* de processadores bidimensionais, onde processadores são interconectados em malhas horizontais e verticais.

Wang, Blough e Alkalaj em [187, 188] apresentam novas condições necessárias e suficientes para um sistema ser t-diagnosticável sobre o diagnóstico baseado em comparações

de ambos o modelo MM e o modelo apresentado por Sengupta e Dahbura. Eles mostram que um sistema é t-diagnosticável se e somente se para todo $Z \subseteq V$ com $Z \neq \emptyset$, e para todo Z_1, Z_2 que particiona Z, $|N_1(Z)| + |N_2(Z)| + CMVC(G_3(Z)) + max(|Z_1| + |Z_2|) > t$, onde: $CMVC(G_3(Z))$ representa a cardinalidade de um conjunto mínimo de cobertura de vértices do $G_3(Z)$; e $N_1(Z) = \{v \in V - Z \mid \exists z \in Z \text{ com } (v, z)_v \in C\}$, isto é, processadores em $V - Z$ que comparam a si próprios com pelo menos um processador em Z; $N_2(Z_1) = \{u \in V - Z - N_1(Z) \mid \exists v, w \in Z_1 \text{ com } (v, w)_u \in C\}$, isto é, processadores em $V - Z - N_1(Z)$ que comparam dois processadores em Z_1; $G_3(Z) = (N_3(Z), E_3(Z))$ tal que $N_3(Z) = \{u \in V - Z - N_1(Z) - N_2(Z) \mid \exists v \in Z \text{ e } w \in V - Z - N_1(Z) - N_2(Z) \text{ com } (v, w)_u \in C \text{ ou } (u, v)_w \in C\}$ e $E_3(Z) = \{\{(u, v)\} \in N_3(Z) \mid \exists w \in Z \text{ com } (v, w)_u \in C\}$. Os autores também apresentam um algoritmo para este modelo e conduzem experimentos através de simulações onde é mostrado que com um número reduzido de testes o algoritmo realiza o diagnóstico do sistema desde que o número de processadores falhos seja relativamente pequeno.

2.4 Algoritmos Polinomiais para Diagnóstico de Sistemas de Topologia Arbitrária sobre o Modelo MM*

Dois algoritmos polinomiais de diagnóstico com base no modelo MM* para sistemas de topologia arbitrária foram propostos na literatura. Sengupta e Dahbura em [169] apresentaram um algoritmo com complexidade $O(N^5)$, onde N é o número de unidades no sistema – e mostram que a diagnosticabilidade de sistemas de topologia arbitrária sobre o modelo MM* é NP-completa. Outro algoritmo de diagnóstico de complexidade $O(N\Delta^3\delta)$ – onde Δ e δ são respectivamente o grau das unidades de maior e menor grau no sistema – foi apresentado por Yang e Tang em [198], também para o modelo MM*. A seguir uma breve descrição destes dois algoritmos de diagnóstico é apresentada.

2.4.1 Um Algoritmo $O(N^5)$ de Diagnóstico Baseado em Comparações

Sengupta e Dahbura em [169] apresentaram um algoritmo polinomial, com complexidade $O(N^5)$, para o diagnóstico de sistemas de topologia arbitrária com N unidades, com base no modelo MM*. O algoritmo determina de forma adaptativa quais comparações serão executadas. Uma unidade i executando este algoritmo inicia seus testes comparando duas unidades $j, k \mid (i,j), (i,k) \in E$, ou seja, a unidade i realiza a comparação $(j,k)_i$. Se a saída da comparação $r((j,k)_i) = 1$ (diferença), a unidade i escolhe um outro par de unidades para comparar, se existir tal par. Se o resultado da comparação $r((j,k)_i) = 0$ (igualdade), então a unidade i usa esta unidade j para comparar *todos* os seus vizinhos, ou seja, todas as comparações $(j,p)_i \mid \forall (p,i) \in E$, com $p \neq j$, são realizadas.

O algoritmo de diagnóstico apresentado por Sengupta e Dahbura – chamado *DIAGNOSIS* – é mostrado na Figura 2.10. O algoritmo recebe como entrada o grafo $G = (V, E)$ e o conjunto com o resultado de todas as comparações, isto é, a síndrome do sistema (σ). Algumas definições são necessárias para entender o algoritmo e são apresentadas abaixo.

Dado um grafo $G' = (V', E')$, $K \subseteq V'$ é um conjunto de vértices de cobertura (*vertex cover set*) de G' se toda aresta em E' for incidente a ao menos um vértice em K. Este conceito pode ser estendido para hipergrafos, permitindo a construção de hiperarestas ao invés de arestas. Um conjunto de cobertura de vértices de menor cardinalidade é chamado de conjunto mínimo de cobertura de vértices. Um subconjunto $M' \subseteq E'$ é chamado de uma *correspondência* se nenhum vértice em V' for incidente a mais de uma aresta em M', e se não formar nenhum ciclo (*self-loop*). Uma correspondência de máxima cardinalidade é chamada de *máxima correspondência*. Conforme a definição apresentada na Seção 2.3.2, um conjunto $X \subset V$ é chamado de um possível conjunto de unidades falhas (*Allowable Fault Set* – AFS) do sistema S para a síndrome σ, se para quaisquer três unidades i, j, k tal que $(i,j)_k \in C$:

```
Algoritmo DIAGNOSIS
/* Entrada: Um sistema MM* t-diagnosticável G = (V, E)
            e uma síndrome de comparações σ */
/* Saída:   O conjunto das unidades falhas */

/* Fase de Inicialização */
F ← ∅;
calcular S(σ);

para cada nodo ∈ S(σ) tal que |N(i)| = t + 1 faça
  /* Primeiro Passo */
  para cada k ∈ N(i) faça
    se N(i) − {k} é um AFS (Allowable Fault Set) então
      F ← N(i) − k;
      termina o algoritmo;
    fim se
  fim para
fim para

para cada nodo ∈ S(σ) tal que |N(i)| = t faça
  /* Segundo Passo */
  para cada k ∈ N(i) faça
    se N(i) é um AFS então
      F ← N(i);
      termina o algoritmo;
    fim se
  fim para

  /* Terceiro Passo */
  calcular H(s);
  para cada k ∈ N(i) faça
    para cada h ∈ H(σ) faça
      se N(i) − k + h é um conjunto de cobertura de
        vértices do hipergrafo Z = (V, H(σ)) então
        F ← N(i) − k + h;
        termina o algoritmo;
      fim se
    fim para
  fim para
fim para

/* Quarto (e Último) Passo */
início
  construir o grafo Y = (V, M(σ));
  remover todos self-loops de Y;
  calcular a maior correspondência de Y;
  F ← o conjunto mínimo de cobertura de vértices de Y;
fim
```

Figura 2.10: O algoritmo *DIAGNOSIS* apresentado por Sengupta e Dahbura.

— se $k \in V - X$ e $i, j \in V - X$ então $r((i,j)_k) = 0$

— se $k \in V - X$ e $\{i,j\} \cap X \neq \emptyset$ então $r((i,j)_k) = 1$

Para a síndrome σ, um AFS de menor cardinalidade é chamado de AFS mínimo de σ – MAFS(σ), e $N(i) = \{j \mid (i,j) \in E\}$ é o conjunto de vizinhos da unidade i. Na fase de inicialização, o conjunto de unidades falhas (F) é atribuído como vazio, e $S(\sigma)$ é calculado. $S(\sigma)$ é o conjunto de comparadores que não retornaram igualdade para todas as comparações executadas.

Em seguida, toda unidade i em $S(\sigma)$ tal que $|N(i)| = t+1$ é examinada. Se removendo uma unidade k de $N(i)$ resultar em um AFS, então o conjunto de unidades falhas $F = N(i) - k$. Se ocorrer este caso o algoritmo então termina.

Caso F não for determinado pelo passo anterior, então toda unidade i em $S(\sigma)$, tal que $|N(i)| = t$, é examinada. Primeiramente, o algoritmo verifica se $N(i)$ é um AFS: se ocorrer este caso, $F = N(i)$, e o algoritmo termina. Caso contrário, ainda existem unidades fora de $N(i)$ que podem estar falhas. Para verificar estas unidades, o hipergrafo $Z = (V, H(\sigma))$ é criado, onde o conjunto $H(\sigma)$ é construído da forma descrita na sequência. Inicialmente $H(\sigma) = \{\{i,j,k\} \mid (i,j)_k \in C \text{ e } r((j,k)_i) = 1\}$. Então o seguinte passo é executado até que $H(\sigma)$ não mude mais: se $\{i,j,k\} \in H(\sigma)$ e m testou k como sem-falha e $\{i,j,m\} \notin H(\sigma)$ então $\{i,j,m\}$ é adicionado ao $H(\sigma)$.

No próximo passo do algoritmo, cada unidade $h \in H(\sigma)$ substitui, uma a uma, cada unidade k em $N(i)$. O algoritmo verifica se o conjunto resultante é uma cobertura de vértices do hipergrafo $Z = (V, H(\sigma))$. Se ocorrer tal caso, o conjunto de unidades falhas foi encontrado, $F = N(i) - k + h$. Neste momento o algoritmo termina.

Finalmente, se F ainda não foi encontrado nos passos anteriores, um novo grafo $Y = (V, M(\sigma))$ é construído, com $M(\sigma)$ construído em cinco passos:

***Passo 1*:** Para qualquer $i \notin S(\sigma)$, se $r((j,k)_i) = 1$ e i testou ambos j e k como sem-falha, então $(i,i) \in M(\sigma)$.

***Passo 2*:** Para qualquer $i \notin S(\sigma)$, se $r((j,k)_i) = 1$ e i testou j como falho,

então $(i,j) \in M(\sigma)$, e se $r((j,k)_i) = 1$ e i testou k como falho, então $(i,k) \in M(\sigma)$.

***Passo 3*:** Para qualquer $i \in S(\sigma)$, se existe $j \in N(i)$ tal que $j \in S(\sigma)$ então $(i,j) \in M(\sigma)$.

***Passo 4*:** Para qualquer $i \in S(\sigma)$, se existe $j \in N(i)$ tal que $j \notin S(\sigma)$ e, se i testou j como sem-falha, então $(j,p) \in M(\sigma)$ para todo $p \in N(i) - \{j\}$, enquanto se i testou j como falho, então $(i,j) \in M(\sigma)$.

***Passo 5*:** Para qualquer $(p,q) \in M(\sigma)$, se p testou x como sem-falha e q testou y como sem-falha o $(x,y) \in M(\sigma)$ e $(p,y) \in M(\sigma)$.

Todos os ciclos são removidos e um algoritmo para o cálculo de composições máximas para grafos gerais, como o algoritmo apresentado em [149], é executado sobre Y. No passo final, o conjunto mínimo de cobertura de vértices F de Y é encontrado usando o algoritmo de diagnóstico baseado no modelo PMC apresentado em [45].

2.4.2 Um Algoritmo $O(N\Delta^3\delta)$ de Diagnóstico Baseado em Comparações

Yang e Tang em [198] apresentam um algoritmo de diagnóstico baseado no modelo MM* com complexidade $O(N\Delta^3\delta)$ para sistemas de topologia arbitrária, onde Δ e δ são respectivamente o grau da unidade de maior e menor grau do sistema. Este algoritmo é uma alternativa ao algoritmo $O(N^5)$ proposto por Sengupta e Dahbura.

O algoritmo envolve não apenas o modelo de diagnóstico baseado em comparações, mas também o modelo de diagnóstico PMC. Inicialmente, a *síndrome de comparações*, isto é, a síndrome que contém o resultado das comparações, é avaliada na tentativa de determinar o conjunto das unidades falhas. Se o diagnóstico não for completado através da síndrome de comparações, então a síndrome é convertida para a *síndrome de testes* do modelo PMC, e um algoritmo clássico de diagnóstico é aplicado para obter o conjunto de unidades falhas. As seguintes definições são necessárias para entender o algoritmo.

Seja σ a síndrome de comparação do sistema. $N(i)$ é o conjunto de vizinhos da unidade i e $d(i) = |N(i)|$ é o grau de i. Para duas unidades adjacentes u e v, v é um filho σ-0 de u se existe $w \in N(u)$ tal que $r((v,w)_u) = 0$, ou seja, v é um filho σ-0 de u se a unidade u avaliar a unidade v como sem-falha; caso contrário, v é um filho σ-1 de u. Uma unidade é um comparador σ-0 se ela possuir ao menos um filho σ-0. Em outras palavras, uma unidade u é um comparador σ-1 se $r((v,w)_u) = 1$ para toda unidade v, w que são comparadas pela unidade u. COMP_1 representa o conjunto de todos os comparadores σ-1. COMP_{10} representa o conjunto de todos os comparadores σ-1 com grau t. COMP_{11} representa o conjunto de todos os comparadores σ-1 com grau $t+1$. COMP_{12} representa o conjunto de todos os comparadores σ-1 com grau $\geq t+2$. $\mathrm{COMP}_1 = \mathrm{COMP}_{10} \cup \mathrm{COMP}_{11} \cup \mathrm{COMP}_{12}$. $\mathrm{SON}_0(u)$ representa o conjunto de todos os filhos σ-0 da unidade u. A unidade u é σ-conflitante se u tem dois filhos σ-0 v e w tal que $r((v,w)_u) = 1$. CONF representa o conjunto de todas as unidades σ-conflitantes.

Uma unidade u é um pai σ-0 de v se existe uma unidade w tal que $r((v,w)_u) = 0$. $\mathrm{PARENT}_0(v)$ representa o conjunto de todas as unidades σ-0, pais da unidade v, e $\mathrm{PARENT}_0(U) = \bigcup_{x \in U} \mathrm{PARENT}_0(x)$. Uma unidade u é um predecessor σ-0 de v se existe a sequência de unidades $w_0 = u, w_1, \ldots, w_p, w_{p+1} = v$ tal que $w_i \in \mathrm{PARENT}_0(w_{i+1})$, para $i = 0, 1, \ldots, p$. $\mathrm{PRED}_0(u)$ representa o conjunto de todos os predecessores σ-0 da unidade u. $\mathrm{PRED}_0(U) = \bigcup_{x \in U} \mathrm{PRED}_0(u)$ e $\mathrm{PRED}_0[U] = \mathrm{PRED}_0(U) \cup U$.

Um conjunto $U \subset V$ é também chamado de *Allowable Fault Set* (AFS) do sistema S, se para quaisquer três unidades u, v, w onde $(v,w)_u \in C$, $u \in V - U$, $v, w \in N(u)$, e $v \neq w$, tal que:

— se $v, w \in U - X$ então $r((v,w)_u) = 0$

— se $\{v, w\} \cap X \neq \emptyset$ então $r((v,w)_u) = 1$

Um t-AFS de σ é um AFS de σ com no máximo t unidades. Seja K um conjunto de unidades tal que $K \subseteq V$, um K^{+1} AFS de σ é um AFS de σ que possui a forma $K \cup \{u\}$ para algum $u \in V - K$. $\mathrm{NODE}^{+1}(K)$ representa o conjunto de todos as unidades K^{+1}

de σ. Um *grupo* K^{+1} de σ é um conjunto de três unidades $u, v, w \in V - K$ tal que, ou $r((v,w)_u) = 1$, ou $r((w,u)_v) = 1$, ou $r((u,v)_w) = 1$. GROUP$^{+1}(K)$ representa o conjunto de todos os *grupos* K^{+1} de σ.

Seja σ a síndrome de comparação do sistema G e H um subsistema de G, a *restrição* de σ sobre H, denotada por $\sigma|_H$, é uma síndrome de comparação sobre H definida por $(v,w)_u$ para todo $u, v, w \in V(H)$, $v, w \in N(u)$, e $v \neq w$, onde $V(H)$ é o conjunto de vértices do grafo H. A síndrome de testes σ induzida sobre o modelo PMC, denotada por $t[\sigma]$, é definida desta forma: para quaisquer duas unidades adjacentes u e v, seja $t[\sigma](u,v) = 0$ ou 1 se, respectivamente, u é um pai σ-0 de v ou não.

O algoritmo de diagnóstico apresentado por Yang e Tang – chamado *MM*_DIAG* – é mostrado na Figura 2.11. A Figura 2.12 mostra o procedimento *CHECK_IF* que é utilizado pelo algoritmo *MM*_DIAG*. O algoritmo recebe como entrada o grafo $G = (V, E)$ que representa o sistema t-diagnosticável e a síndrome de comparação do sistema. O algoritmo produz como saída o conjunto de unidades falhas.

O algoritmo é dividido em três fases. Na primeira fase o algoritmo define os conjuntos COMP_{10}, COMP_{11} e COMP_{12}, ou seja, identifica todos os comparadores σ-1. Os autores provam que todos os comparadores no conjunto COMP_{12} estão falhos. Para todo comparador x nos conjuntos COMP_{11} ou COMP_{10}, o algoritmo verifica todos os possíveis candidatos para um t-AFS desde que x esteja sem-falha.

Neste contexto o procedimento *CHECK_IF* é importante. Este procedimento recebe como entrada um sistema $G = (V, E)$, a síndrome de comparação correspondente σ, e um conjunto $K \subset V$ que não é um AFS. O procedimento retorna um outro conjunto $K^{+1} = K \cup \{u\} \mid u \in V - K$, se existir tal conjunto. Este novo conjunto precisa ser um AFS e precisa ter $|K| + 1$ unidades. Caso contrário, o procedimento retorna o valor "No". Se um t-AFS for encontrado em um destes passos, o diagnóstico está completo e o conjunto AFS encontrado é retornado como o conjunto de unidades falhas. Caso contrário, todos os comparadores COMP_1 são considerados falhos e o algoritmo passa para a segunda fase.

Algoritmo: MM*_DIAG
/* *Entrada*: Um sistema MM* t-diagnosticável $G = (V, E)$
e uma síndrome de comparações σ */
/* *Saída*: O conjunto das unidades falhas */

início
/* *Primeira Fase* */
para cada nodo u de V **faça** determinar $\text{PARENT}_0(u)$ e $\text{SON}_0(u)$;
calcular COMP_1, COMP_{10} e COMP_{11};
se existir $u \in \text{COMP}_{11}$ e $v \in N(u)$ tal que
$N(u) - \{v\}$ é um AFS de σ **então**
retornar $(N(u) - \{v\})$;
fim se
se existir $u \in \text{COMP}_{10}$ tal que $N(u)$ é um AFS de σ **então**
retornar $(N(u))$;
fim se
se existir $u \in \text{COMP}_{10}$ e $v \in N(u)$ tal que
$N(u) - \{v\}$ é um AFS de σ **então**
retornar $(N(u) - \{v\})$;
fim se
se existir $u \in \text{COMP}_{10}$ e $v \in N(u)$ tal que
CHECK_IF(G, $N(u) - \{v\}, \sigma) \neq$ "No.")
retornar (CHECK_IF(G, $N(u) - \{v\}, \sigma)$);
end if
/* *Segunda Fase* */
determinar CONF;
$U \leftarrow \text{PRED}_0[\text{COMP}_1 \cup \text{CONF}]$;
/* *Terceira Fase* */
construir o subgrafo $H = G - U$, construir a síndrome de testes $t[\sigma|_H]$;
encontrar o menor AFS U' de $t[\sigma|_H]$ através do algoritmo de Sullivan;
retornar $(U \cup U')$;
fim

Figura 2.11: O algoritmo de diagnóstico *MM*_DIAG* apresentado por Yang e Tang.

```
Procedimento: CHECK_IF
/* Entrada: Um sistema G = (V, E),
            A síndrome de comparações σ, e
            Um conjunto K ⊂ V que não é um AFS de σ */
/* Saída:   Um K^{+1} AFS de σ caso exista, ou "No" caso não exista */

início
   calcular NODE^{+1}(K);
   calcular GROUP^{+1}(K);
   se |NODE^{+1}(K)| ≥ 2 então
         retornar ("No");
   fim se
   se |NODE^{+1}(K)| = 1 e
         se K ∪ NODE^{+1}(K) é um AFS de σ então
               retornar (K ∪ NODE^{+1}(K));
         senão
               retornar ("No");
         fim se
   fim se
   se |NODE^{+1}(K)| = 0 então
         se ∪_{c∈GROUP^{+1}(K)} c = Φ então
               retornar ("No");
         senão
               se existir u ∈ ∪_{c∈GROUP^{+1}(K)} c tal que K ∪ {u}
                     é um AFS de σ então
                     retornar (K ∪ {u});
               senão retornar ("No");
               fim se
         fim se
   fim se
fim
```

Figura 2.12: O procedimento *CHECK_IF* utilizado pelo algoritmo *MM*_DIAG*.

Na segunda fase, o conjunto CONF é identificado, isto é, todas as unidades que possuem dois filhos σ-0 v e w, mas com comparação $r((v,w)_u) = 1$. Os autores provam que todas as unidades do conjunto CONF estão falhas. Então, o algoritmo define um novo conjunto PRED_0 com base nos conjuntos COMP_1 e CONF. O conjunto $U = \text{PRED}_0[\text{COMP}_1 \cup \text{CONF}]$ representa todos os predecessores que testaram diretamente ou indiretamente alguma unidade nos conjuntos COMP_1 e CONF. Os autores também provam que todas as unidades em $\text{PRED}_0[\text{COMP}_1 \cup \text{CONF}]$ estão falhas.

Na terceira e última fase do algoritmo, a tarefa de diagnóstico é convertida para uma tarefa do modelo PMC. Um subconjunto $H = G - U$ composto de todas as unidades que ainda não foram identificadas como falhas nas duas etapas anteriores é construído. Em um passo chave do algoritmo, uma síndrome de testes $t[\sigma|_H]$ baseada nas unidades de H é construída a partir da síndrome de comparação original σ. Os autores então provam que H é $(t-|U|)$-diagnosticável sobre o modelo PMC e que $F-U$ é o único conjunto $(t-|U|)$-AFS possível dada a síndrome de testes $t[\sigma|_H]$. Portanto a localização das unidades falhas remanescentes é equivalente a encontrar o AFS mínimo da síndrome de testes. Então é possível encontrar o conjunto AFS mínimo aplicando o algoritmo $O(\delta^3 + |E|)$ apresentado por Sullivan em [177].

2.5 Modelos Generalizados de Diagnóstico Distribuído Baseado em Comparações

O modelo generalizado de diagnóstico distribuído baseado em comparações [6, 5] assume um sistema S completamente conectado também representado por um grafo $G = (V, E)$, no qual $\forall i \in V$ e $\forall j \in V$, $\exists\ (i,j) \in E$. Este modelo incorpora além de todas as asserções do modelo MM, mais uma: o tempo para um nodo sem-falha produzir e transmitir a saída para uma tarefa é limitado. Este modelo é completamente distribuído, ou seja, nodos sem-falha executam as comparações e também realizam o diagnóstico do sistema baseado na síndrome de comparações.

O modelo generalizado define um multigrafo, $M(S)$, para representar a maneira como os testes são executados no sistema. $M(S)$ é um multigrafo direcionado definido sobre o grafo G, quando todos os nodos do sistema são sem-falha. Os nodos do sistema podem estar falhos ou sem-falha. Um nodo se torna falho quando ele deixa de funcionar ou ainda quando retorna saídas arbitrárias para uma determinada tarefa. Uma mudança de estado de um nodo é chamada de um *evento*. Os estados dos nodos são também determinados através da comparação da saída da tarefa de um nodo com a saída gerada por outro nodo para a mesma tarefa. Este modelo também assume que: o resultado da comparação realizada por um nodo sem-falha sobre as saídas produzidas por um nodo falho e qualquer outro nodo falho ou sem-falha, sempre indica diferença.

Em [6, 5] um algoritmo de diagnóstico hierárquico adaptativo e distribuído em nível de sistema baseado em comparações – chamado *Hi-Comp* – é apresentado e é baseado neste modelo generalizado. O algoritmo é distribuído, isto é, executa em todo nodo do sistema e todo nodo realiza o diagnóstico completo. Uma *rodada de testes* é definida como o intervalo de tempo que todos os nodos sem-falha precisam para diagnosticar todos os nodos do sistema. Uma asserção é feita onde após o nodo i testar o nodo j em uma certa rodada de testes, o nodo j não pode sofrer nenhum novo evento naquela rodada de testes.

O algoritmo emprega um conjunto de testes representado por um grafo $T(S)$ que é um hipercubo virtual quando o número de nodos é uma potência de dois. Os nodos possuem identificadores sequenciais $(0..N-1)$, e cada nodo pode então determinar o conjunto de vizinhos em $T(S)$. A *distância de diagnóstico* entre o nodo i e o nodo j, $d_{i,j}$, é definida como a menor distância entre o nodo i e o nodo j em $T(S)$. Por exemplo, na Figura 2.13 a distância de diagnóstico entre o nodo 0 e o nodo 2 é 1.

Um grafo $T_i(S)$ é definido como o grafo direcionado baseado na $T(S)$ e mantido pelo nodo i. Este grafo mostra como os nodos do sistema obtêm informações de diagnóstico. A Figura 2.13 mostra $T_0(S)$ para um sistema de 8 nodos; o nodo 0 obtém informações de diagnóstico sobre (a) os nodos $[3, 5, 7]$ a partir do nodo 1, (b) os nodos $[3, 6, 7]$ a partir do nodo 2, e (c) os nodos $[5, 6, 7]$ a partir do nodo 4.

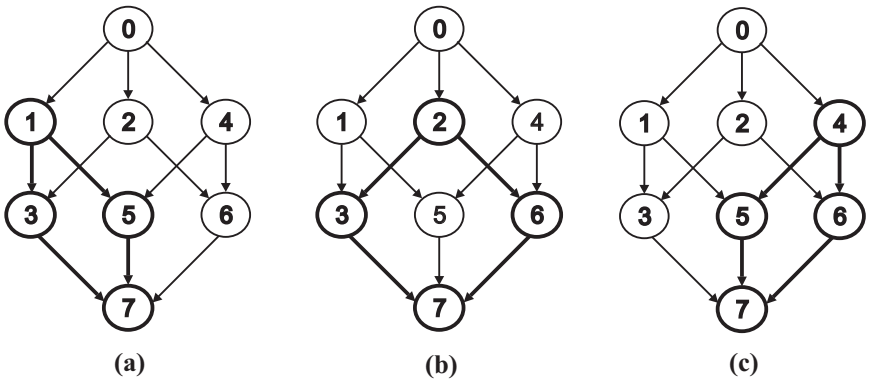


Figura 2.13: $T_0(S)$: o nodo 0 obtém informações de diagnóstico a partir dos nodos 1, 2 e 4.

Em cada rodada de testes, um nodo i executando o algoritmo *Hi-Comp* inicialmente testa os seus filhos na $T_i(S)$ em pares. Quando a comparação de dois nodos diferentes p e q indicar igualdade, o nodo i classifica estes dois nodos testados como sem-falha. Caso contrário, se a comparação indicar diferença, os nodos testados são classificados inicialmente como *indefinidos*. Neste momento, se o nodo i já tiver identificado qualquer par de nodos como sem-falha, então ele compara um dos nodos sem-falha com cada um dos dois nodos indefinidos. Por outro lado, se o nodo i ainda não tiver identificado nenhum nodo sem-falha, os dois nodos permanecem indefinidos. Se após o nodo i testar todos os seus filhos, ainda não existir nenhum nodo sem-falha identificado, isto é, todos os seus filhos são classificados como indefinidos, o nodo i então realiza testes nos filhos dos seus filhos, e assim por diante, até que uma comparação resulte em igualdade ou até que o nodo i realize testes em todos os nodos da $T_i(S)$.

Assim que um nodo i classificar qualquer nodo p como sem-falha, este nodo i obtém a partir do nodo p informações de diagnóstico sobre todo nodo $k \in V \mid d_{i,k} \leq d_{i,p} + d_{p,k}$. Um nodo i pode obter informação de diagnóstico sobre um nodo j através de mais de um nodo. Como exemplo, na Figura 2.13 o nodo 0 pode obter informação de diagnóstico sobre o nodo 3 a partir do nodo 1 ou a partir do nodo 2. Para assegurar que o nodo i sempre receberá a informação de diagnóstico mais recente sobre um determinado nodo j o algoritmo utiliza *contadores de tempo* (*timestamps*), que são implementados através de contadores de eventos [54].

A latência de diagnóstico do algoritmo *Hi-Comp* é provada pelos autores como sendo igual a $log_2 N$ rodadas de testes no pior caso, o número máximo de testes executados é $O(N^3)$, e o algoritmo é $(N-1)$-diagnosticável.

Outro modelo generalizado de diagnóstico distribuído e adaptativo baseado em comparações foi proposto em [208]. Neste modelo um nodo sem-falha testa outros nodos, e com base no resultado dos testes o testador classifica os nodos testados em conjuntos. Um teste também é realizado através do envio de uma tarefa a dois nodos diferentes. As saídas das tarefas são então comparadas; se a comparação resultar em igualdade, os dois nodos são classificados no mesmo conjunto. Por outro lado, se a comparação resultar em diferença, os dois nodos são classificados em conjuntos diferentes, de acordo com a saída da tarefa. Um dos conjuntos contém todos os nodos sem-falha do sistema. Se os nodos forem classificados em mais de um conjunto, então pode-se dizer que existem nodos falhos no sistema.

Este modelo generalizado identifica nodos falhos que pararam de funcionar, e também identifica nodos que continuam funcionando, mas não retornam a saída correta e esperada para uma determinada tarefa de entrada. As seguintes asserções são feitas sobre o sistema:

1. a comparação realizada por um nodo sem-falha sobre as saídas de tarefas produzidas por dois nodos também sem-falha sempre resulta em igualdade;
2. a comparação realizada por um nodo sem-falha sobre as saídas de tarefas produzidas por um nodo falho e outro nodo sem-falha sempre resulta em diferença; e,
3. o intervalo de tempo requerido para um nodo sem-falha produzir a saída de uma tarefa é limitado.

O modelo proposto é o primeiro modelo distribuído baseado em comparações que permite que as saídas de dois nodos falhos sejam iguais, ou seja, a comparação das saídas

geradas por dois nodos falhos pode resultar em igualdade. Esta é uma característica também presente no modelo centralizado de Chwa e Hakimi [42].

Em [208] os autores também propõem um algoritmo, chamado *Hi-Dif*, para este modelo generalizado. O algoritmo também emprega uma estratégia de testes representada por um grafo $T(S)$ que é um hipercubo virtual. O algoritmo identifica nodos falhos que param e não funcionam mais, e também classifica os demais nodos em conjuntos. Estes conjuntos permitem a identificação de quais nodos retornam uma dada saída para a tarefa de entrada. Com esta configuração de conjuntos é possível identificar quais são os nodos sem-falha, nodos falhos que pararam de funcionar, e também os nodos que retornaram saídas diferentes da considerada correta para as tarefas enviadas. É possível também identificar quantas são as diferentes saídas retornadas e quem são os nodos que retornaram cada uma destas diferentes saídas. Prova-se que a latência do algoritmo *Hi-Dif* é $log_2 N$ rodadas de testes no pior caso, o número máximo de testes executados é $O(N^2)$, e que o algoritmo é $(N-1)$-diagnosticável.

CAPÍTULO 3

UM NOVO ALGORITMO DE DIAGNÓSTICO BASEADO EM COMPARAÇÕES PARA SISTEMAS DE TOPOLOGIA ARBITRÁRIA

Este capítulo apresenta um novo algoritmo de diagnóstico baseado em comparações para a identificação das unidades falhas em sistemas t-diagnosticáveis de topologia arbitrária, com base no modelo MM*. O algoritmo é completo e correto, ou seja, prova-se que o algoritmo identifica corretamente o estado de todas as unidades falhas do sistema. Este novo algoritmo é uma alternativa que possui complexidade significativamente menor que a dos algoritmos previamente apresentados por Sengupta e Dahbura [169] e por Yang e Tang [198]. Além disso, a solução proposta é também a primeira a realizar o diagnóstico de sistemas de topologia arbitrária com base apenas na síndrome de comparações. Ambos os algoritmos previamente publicados aplicam técnicas que convertem a síndrome de comparações para uma síndrome de testes do modelo PMC [158], para que o diagnóstico seja realizado.

Antes de apresentar detalhes do novo algoritmo, é importante recordar algumas definições do modelo MM e do modelo MM*, além das condições de diagnosticabilidade do sistema. Estes detalhes já foram apresentados na Seção 2.3, mas são registrados novamente de forma resumida a seguir, na Seção 3.1; esta Seção 3.1 ainda apresenta importantes definições e notações que são usadas pelo algoritmo de diagnóstico proposto. Na sequência, a Seção 3.2 descreve o algoritmo de diagnóstico proposto. A Seção 3.3 apresenta as provas de correção e análise de complexidade. Por fim, a Seção 3.4 apresenta os resultados experimentais.

3.1 Definições Preliminares

O modelo MM [140] representa um sistema S com N unidades através de um grafo $G = (V, E)$, onde V é o conjunto de unidades e E é o conjunto de enlaces de comunicação entre as unidades. O conjunto de todas as unidades falhas é representado por F. Para uma unidade $i \in V$, $N(i) = \{j \mid (i, j) \in E\}$ denota o conjunto de unidades vizinhas de i e $d(i) = |N(i)|$ é a ordem – ou o grau – da unidade i, isto é, ambos $d(i)$ e $|N(i)|$ correspondem ao número de vizinhos de i.

Os testes realizados no sistema são representados por um multigrafo $M = (V, C)$, onde V também é o conjunto das unidades e C representa o conjunto de todos os testes (ou comparações) realizadas no sistema. Cada comparação $(j, k)_i \in C$ representa um teste, onde a unidade testadora (ou comparadora) i envia uma mesma tarefa para as unidades j e k, e compara a saída das tarefas recebidas de ambas as unidades.

A notação $r((j, k)_i)$ representa o resultado da comparação $(j, k)_i$. O resultado é 0 – ou seja, $r((j, k)_i) = 0$ – quando a comparação resulta igualdade, e o resultado é 1 – ou seja, $r((j, k)_i) = 1$ – quando a comparação indica qualquer diferença. Os resultados das comparações são enviados ao observador central, uma entidade confiável que realiza o diagnóstico completo do sistema. O conjunto com o resultado de todas as comparações realizadas no sistema é chamado de síndrome do sistema – ou síndrome de comparações – e representado por σ. Em outras palavras, σ é o conjunto com todos $r((j, k)_i)$ tal que $(j, k)_i \in C$.

As principais asserções sobre os resultados de testes no modelo MM são:

— A comparação realizada por uma unidade sem-falha sobre as saídas de tarefas executadas e retornadas por duas unidades também sem-falha sempre resulta em igualdade.

— A comparação realizada por uma unidade sem-falha sobre as saídas de tarefas de uma unidade falha e qualquer outra unidade (falha ou sem-falha) sempre resulta em diferença.

— A comparação realizada por uma unidade falha, independente do estado das unidades comparadas, não é confiável, e pode resultar tanto em igualdade como em diferença.

O modelo MM* também apresentado em [140] e é um caso especial do modelo MM, ou seja, o modelo MM abrange o modelo MM*. A diferença é que no modelo MM* uma unidade testadora executa comparações para todas as suas unidades vizinhas, em pares.

Um sistema é t-diagnosticável se todas as unidades falhas do sistema puderem ser identificadas desde que o número de unidades falhas não seja maior que t. Sengupta e Dahbura [169] apresentam as condições necessárias para um sistema ser t-diagnosticável sobre o modelo MM, que seguem abaixo.

Sejam $S_1, S_2 \subset V$, considere o conjunto $\sigma(S_1)$ como o conjunto de síndromes que podem ser produzidas se S_1 é o conjunto de nodos falhos, e considere o conjunto $\sigma(S_2)$ analogamente definido. Sengupta e Dahbura provam que um sistema S com N unidades é t-diagnosticável se e somente se para cada par de conjuntos $S_1, S_2 \subset V$ tal que $S_1 \neq S_2$ e $|S_1|, |S_2| \leq t$, $\sigma(S_1) \cap \sigma(S_2) = \emptyset$. Em outras palavras, um sistema é t-diagnosticável se para cada síndrome existe um conjunto único de unidades falhas que pode produzir aquela síndrome, desde que o número de unidades falhas não seja maior que t.

Os autores também provam que para um sistema com N unidades ser t-diagnosticável, $N \geq 2t + 1$ e cada nodo deve possuir grau maior ou igual a t, isto é, para qualquer $i \in V$, $|N(i)| \geq t$.

Sengupta e Dahbura também definem um conjunto $U \subset V$ como um AFS (*Allowable Fault Set*) para a síndrome σ, se para quaisquer três unidades i, j, k tal que $(j, k)_i \in C$, as seguintes condições são satisfeitas:

— se $i \in V - U$ e $j, k \in V - U$ então $r((j, k)_i) = 0$

— se $i \in V - U$ e $\{j, k\} \cap U \neq \emptyset$ então $r((j, k)_i) = 1$

Denota-se por t-AFS um conjunto AFS com no máximo t unidades. Sengupta e Dahbura destacam que como em um sistema t-diagnosticável o conjunto de unidades

falhas F é único e $|F| \leq t$, então existe apenas um único conjunto AFS como no máximo t unidades, isto é, existe apenas um t-AFS no sistema.

3.1.1 Definições e Notações Usadas Pelo Algoritmo

A seguir são apresentadas importantes definições e notações que são usadas pelo algoritmo de diagnóstico proposto para sistemas de topologia arbitrária.

Definição 1: Um *caminho* (ou *path*) em G, $P[v_0, v_z] = \langle v_0, v_i, \ldots, v_z \rangle$ onde $\{v_0, v_i, \ldots, v_z\} \subseteq V$, é uma sequência de vértices distintos tal que qualquer par de vértices consecutivos são adjacentes; v_0 e v_z são respectivamente os vértices inicial e final do caminho.

Definição 2: $G' = (V', E')$ é um subgrafo de $G = (V, E)$ *induzido por* V', denotado por $G[V']$, se $E' = \{(u, v) \in E \mid u, v \in V'\}$.

Definição 3: Seja um *componente conexo máximo* (ou *maximal connected component*) de um grafo não direcionado $G = (V, E)$, um subgrafo $G_x = (V_x, E_x)$ onde $V_x \subseteq V$, $E_x = \{(j, k) \in E \mid j, k \in V_x\}$ tal que qualquer par de vértices $v_a, v_b \in V_x$ são conectados um ao outro por pelo menos um caminho $P[v_a, v_b]$ e não existe nenhum par de vértices v_x, v_y tal que $v_x \in V_x$, $v_y \in V - V_x$ e existe a aresta $(v_x, v_y) \in E$.

Neste trabalho um componente conexo máximo de um grafo G é chamado simplesmente de um componente de G.

Definição 4: Considerando o grafo $G = (V, E)$, seja $\overline{G_Z} = (V - Z, \overline{E_Z})$ o subgrafo resultante da remoção de um conjunto de vértices Z de V, $Z \subset V$ e $\overline{E_Z} = \{(j, k) \in E \mid j, k \in V - Z\}$. $\overline{G_Z}$ pode conter mais de um componente conexo.

Definição 5: $\xi(G)$ e $|\xi(G)|$ representam respectivamente o conjunto e o número de componentes maximais do grafo G. Além disso, $\xi_m(G)$ representa um dos componentes maximais de G, e $1 \leq m \leq |\xi(G)|$.

Definição 6: Seja o conjunto FF_i (*Fault-Free set as seen by unit* i, ou conjunto das unidades sem-falha pela visão da unidade i), $1 \leq i \leq N$, definido como segue: se $r((j,k)_i) = 0$ então $j, k \in FF_i$.

Por esta definição, FF_i é o conjunto das unidades onde pelo menos uma das comparações realizadas pela unidade testadora i resultaram em igualdade. Se a unidade i for sem-falha, o conjunto FF_i contém as unidades sem-falha que são vizinhas de i.

Definição 7: Seja o conjunto F_i (*Falty set as seen by unit* i, ou conjunto das unidades falhas pela visão da unidade i), $1 \leq i \leq N$, definido como segue: se $r((j,k)_i) = 1$ e $k \in FF_i$ então $j \in F_i$.

Por esta definição, F_i é o conjunto de unidades j tal que pelo menos uma das comparações executadas pela unidade i sobre j e qualquer outra unidade $k \in FF_i$ resultaram em diferença. Se a unidade i está sem falha e $|FF_i| > 0$, então o conjunto F_i contém as unidades falhas que são vizinhas de i.

Note que se a unidade comparadora i está sem falha, então $FF_i \cap F_i = \emptyset$, caso contrário, se a comparadora i é uma unidade falha, suas comparações podem ser inconsistentes e uma situação onde $FF_i \cap F_i \neq \emptyset$ pode ocorrer.

Definição 8: O conjunto $FF_i^{\diamond}$, $1 \leq i \leq N$, é definido como segue: se $i \in FF_v$ então $v \in FF_i^{\diamond}$.

Definição 9: O conjunto $F_i^{\diamond}$, $1 \leq i \leq N$, é definido como segue: se $i \in F_v$ então $v \in F_i^{\diamond}$.

A partir das duas definições acima, os conjuntos $F_i^{\diamond}$ e $FF_i^{\diamond}$ representam as unidades comparadoras que consideram a unidade i como sendo falha e sem-falha, respectivamente.

Definição 10: $CompFF_{i,j}$ é um conjunto com três unidades $\{i, j, k\}$ tal que $\exists r((j,k)_i) = 0$.

Definição 11: $CompF_{i,j}$ é um conjunto com três unidades $\{i, j, k\}$ tal que $\exists r((j,k)_i) = 1$ e $k \in FF_i$.

Em outras palavras, dada qualquer uma das comparações realizadas pela unidade i tal que $r((j,k)_i) = 0$, $CompFF_{i,j}$ é o conjunto das três unidades desta comparação, isto é, $CompFF_{i,j} = \{i, j, k\}$.

De forma análoga, $CompF_{i,j}$ representa as três unidades em qualquer uma das comparações realizadas por i onde $r((j,k)_i) = 1$ e $k \in FF_i$.

Definição 12: O conjunto P_i (*Pending set as seen by unit* i, ou conjunto das unidades pendentes pela visão da unidade i), $1 \leq i \leq N$, é definido como segue: se $\nexists r((j,k)_i) = 0$ onde $j, k \in N(i)$ então $P_i = N(i)$, caso contrário $P_i = \emptyset$.

Esta definição indica que P_i contém todos os vizinhos da unidade comparadora i somente quando todas as comparações realizadas por i resultaram em diferença. Em outras palavras, não é possível concluir qualquer coisa sobre o estado das unidades vizinhas de i usando apenas estas comparações realizadas por i. Esta situação ocorre quando $FF_i = \emptyset$ (e consequentemente também $F_i = \emptyset$). Se ao menos uma das comparações executadas por i resultar em igualdade, então é possível notar que todos os vizinhos de i são inseridos em um dos conjuntos FF_i ou F_i, isto é, $FF_i \cup F_i = N(i)$. Neste caso $P_i = \emptyset$.

Definição 13: O conjunto FF'_i é definido como segue: i está sempre em FF'_i; $j \in FF'_i$ se existe pelo menos um caminho $P[i,j]$ da unidade i para a unidade j tal que para todo par de vértices distintos e consecutivos (v_1, v_2) em $P[i,j]$, $v_2 \in FF_{v_1}$.

Em outras palavras, se a unidade i está sem-falha, FF'_i contém i e também toda unidade sem-falha j para as quais existe um caminho $P[i,j]$, caminho este que consiste apenas de unidades sem-falha. Além disso se i está sem-falha então $|FF'_i| > 1$ se e somente se pelo menos duas unidades vizinhas de i estão sem-falha, ou seja, existe ao menos uma comparação realizada por i que resultou em igualdade.

Definição 14: O conjunto F'_i é definido como segue: $\forall u \in FF'_i, F'_i \leftarrow F'_i \cup F_u$.

Se i é uma unidade sem-falha, a Definição 14 implica que o conjunto F'_i contém todas as unidades falhas que são vizinhas de qualquer unidade em FF'_i. Além disso, existe um

caminho em G partindo de qualquer unidade sem-falha em FF'_i para qualquer unidade falha $u \in F'_i$, que consiste apenas de unidades sem-falha com exceção do vértice final u.

É importante destacar que para qualquer unidade sem-falha $i \in V$, se $|FF'_i| > 1$ e existe pelo menos uma unidade falha no sistema, então $|F'_i| > 0$. Além disso, se i está sem-falha e todas as unidades falhas são vizinhas de ao menos uma unidade de FF'_i, então F'_i é o conjunto real de unidades falhas F do sistema. Neste caso é fácil de visualizar que FF'_i é um AFS.

Definição 15: Um conjunto F'_i é definido como *máximo* se i está sem-falha e $\forall j \in V$, $j \neq i$, $|F'_j| \leq |F'_i|$.

Através desta definição, se i está sem-falha e F'_i é máximo, existe um caminho – que consiste apenas de unidades sem-falha com exceção do vértice final – da unidade i para o maior conjunto de unidades falhas.

Definição 16: Seja $\mathcal{S}$ (*Suspect set*, ou conjunto de unidades suspeitas) um conjunto que consiste das três unidades $\{s_1, s_2, s_3\}$ envolvidas em uma comparação $(s_2, s_3)_{s_1} \in C$, tal que uma das seguintes duas condições não são satisfeitas quando se verifica se um conjunto U é um AFS:

— se $s_1 \in V - U$ e $s_2, s_3 \in V - U$ então $r((s_2, s_3)_{s_1}) = 0$;

— se $s_1 \in V - U$ e $\{s_2, s_3\} \cap U \neq \emptyset$ então $r((s_2, s_3)_{s_1}) = 1$.

Note que as duas condições acima são exatamente as condições originais usadas para verificar se um dado conjunto U é um AFS. Em outras palavras, por esta definição, o conjunto suspeito $\mathcal{S} = \{s_1, s_2, s_3\}$ contém as três unidades envolvidas em qualquer comparação $(s_2, s_3)_{s_1} \in C$ que não satisfaz uma das condições de verificação do AFS.

3.2 O Algoritmo de Diagnóstico para Sistemas de Topologia Arbitrária

Nesta seção, o novo algoritmo de diagnóstico para identificação de falhas em sistemas t-diagnosticáveis de topologia arbitrária é descrito e especificado. Antes de apresentar o algoritmo propriamente dito, uma nova estratégia eficiente para verificar se um determinado conjunto U é um AFS (*Allowable Fault Set*) também é apresentada neste trabalho, e é descrita a seguir.

3.2.1 A Função is_AFS

A nova função is_AFS proposta neste trabalho tem o objetivo de determinar se um determinado conjunto de unidades é um AFS. Antes de apresentar a função is_AFS propriamente dita, é importante lembrar que, como definido por Sengupta e Dahbura, um conjunto $U \subset V$ é um AFS se para quaisquer três unidades i, j, k tal que $(j,k)_i \in C$, $j, k \in N(i)$ e $j \neq k$, as seguintes duas condições são satisfeitas:

— se $i \in V - U$ e $j, k \in V - U$ então $r((j,k)_i) = 0$

— se $i \in V - U$ e $\{j,k\} \cap U \neq \emptyset$ então $r((j,k)_i) = 1$

A função is_AFS proposta é apresentada na Figura 3.1. A função recebe como entrada um conjunto $U \subset V$, e retornar "true" se o conjunto U for um AFS, e "false" no caso contrário. A função assume que todos os conjuntos FF_i, F_i e P_i já foram previamente calculados. Prova-se que a função is_AFS determina corretamente quando um dado conjunto $U \subset V$ é um AFS no Teorema 1, que é apresentado na próxima seção deste capítulo. A função executa três verificações: as duas primeiras verificações (nas linhas 4–9 e 11–16) consideram a situação na qual para um dado testador i, $P_i = \emptyset$, isto é, $FF_i \neq \emptyset$; a terceira verificação (linhas 18–29) consideram o caso no qual $P_i \neq \emptyset$.

Com base nas condições apresentadas por Sengupta e Dahbura para verificar o AFS, a função is_AFS começa verificando cada testador $i \in V - U$ (linha 2). Nas linhas 4–9

Função: is_AFS
/* *Entrada*: um conjunto U */
/* *Saída*: $True/False$ e (opcionalmente) o *array* de unidades suspeitas $\mathcal{S}$[1,2,3] */

```
 1: início
 2:   para cada i ∈ V − U faça
 3:     S[1,2,3] ← null;
 4:     para cada j ∈ FF_i faça
 5:         se j ∈ U então
 6:             S[1,2,3] ← CompFF_{i,j};
 7:             retornar False, S[1,2,3];
 8:         fim se
 9:     fim para
10:
11:     para cada j ∈ F_i faça
12:         se j ∈ V − U então
13:             S[1,2,3] ← CompF_{i,j};
14:             retornar False, S[1,2,3];
15:         fim se
16:     fim para
17:
18:     count ← 0;
19:     para cada j ∈ P_i faça
20:         se j ∈ V − U então
21:             count ← count + 1;
22:             se S[1,2] = [null, null] então S[1,2] ← [i,j];
23:             senão se S[3] = null então S[3] ← j;
24:             fim se
25:         fim se
26:     fim para
27:     se count ≥ 2 então
28:         retornar False, S[1,2,3];
29:     fim se
30:   fim para
31:   retornar True;
32: fim
```

Figura 3.1: Código da função is_AFS.

a função verifica cada unidade $j \in FF_i$ para as quais existe pelo menos uma comparação onde $r((j,k)_i) = 0$. Se $j \in FF_i$ e $j \in U$ então a função retorna "false". De forma análoga, nas linhas 11–16 a função verifica cada unidade $j \in F_i$, para as quais existe pelo menos uma comparação onde $r((j,k)_i) = 1$ e $k \in V - U$. Se $j \in F_i$ e $j \in V - U$ a função retorna "false".

Já nas linhas 18–29, a função verifica cada unidade $j \in P_i$, isto é, ela verifica cada unidade vizinha de i tal que todas as comparações realizadas por i resultaram em diferença. Se existir pelo menos duas unidades $j', j'' \in N(i)$ tal que $j', j'' \in V - U$ então, como $j', j'' \in P_i$ e portanto $r((j', j'')_i) = 1$, a função retorna "false". Por fim, se nenhuma das três situações acima acontecem, a função retorna "true" (linha 31), isto é, a função indica que o conjunto U é um AFS. Uma importante observação final é que a função pode opcionalmente retornar o conjunto de unidades suspeitas $\mathcal{S}$ correspondente, caso o conjunto U não seja um AFS.

3.2.2 O Algoritmo de Diagnóstico

O novo algoritmo de diagnóstico *Diag* é apresentado na Figura 3.2. O algoritmo recebe como entrada dois parâmetros: o grafo $G = (V, E)$ representando um sistema MM* t-diagnosticável, e a síndrome de comparações σ correspondente. A saída do algoritmo é o conjunto das unidades falhas F.

Inicialmente o algoritmo (na linha 3) calcula os conjuntos FF_i, F_i, P_i, $FF_i^{\diamond}$ e $F_i^{\diamond}$. O algoritmo é organizado em duas fases, descritas a seguir.

Fase 1. A primeira fase do algoritmo de diagnóstico possui um propósito bem específico: garantir que o algoritmo irá avançar para a segunda fase somente se todas as unidades sem-falha do sistema possuem pelo menos outras duas unidades sem-falha como vizinhas, isto é, para cada unidade sem-falha i, $|FF_i| > 0$. Em outras palavras, caso exista alguma unidade sem-falha i tal que $|FF_i| = 0$, a segunda fase não será executada e portanto esta primeira fase é a fase que deve realizar o diagnóstico do sistema, ou seja, encontrar o conjunto das unidades falhas.

Algoritmo: *Diag*
/* *Entrada*: Um sistema MM* t-diagnosticável $G = (V, E)$ e uma síndrome de comparações */
/* *Saída*: O conjunto das unidades falhas F */

início
  /* *Inicialização* */
  $\forall i \in V$ calcular $FF_i, F_i, P_i, FF_i^{\diamond}, F_i^{\diamond}$;

  /* *Primeira Fase – verifica unidades cujas comparações retornaram apenas diferença* */
  para cada $i \in V$ tal que $|P_i| = t + 1$ **faça**
    para cada $j \in P_i$ **faça**
      se is_AFS($P_i - \{j\}$) **então** retornar $P_i - \{j\}$; **fim se**
    fim para
  fim para

  para cada $i \in V$ tal que $|P_i| = t$ **faça**
    se is_AFS(P_i) **então** retornar P_i; **fim se**
    para cada $j \in P_i$ **faça**
      se is_AFS($P_i - \{j\}, \mathcal{S}$[1,2,3]) **então**
        retornar $P_i - \{j\}$;
      senão
        se is_AFS($P_i - \{j\} \cup \mathcal{S}$[1]) **então**
          retornar $P_i - \{j\} \cup \mathcal{S}$[1];
        senão se is_AFS($P_i - \{j\} \cup \mathcal{S}$[2]) **então**
          retornar $P_i - \{j\} \cup \mathcal{S}$[2];
        senão se is_AFS($P_i - \{j\} \cup \mathcal{S}$[3]) **então**
          retornar $P_i - \{j\} \cup \mathcal{S}$[3];
        fim se
      fim se
    fim para
  fim para

  /* *Segunda Fase – identifica as unidades falhas, caso o AFS não foi encontrado na Fase 1* */
  para cada $i \in V$ **faça**
    $auxFF \leftarrow \emptyset$; $auxF \leftarrow \emptyset$; $CheckedSet \leftarrow \emptyset$;
    calcular FF_i' e F_i';
    $auxFF \leftarrow FF_i'$;
    $auxF \leftarrow F_i'$;
    $CheckedSet \leftarrow FF_i'$;

    se $|F_i'| < t$ **então**
      para cada $j \in V \mid j \notin \{auxF \cup auxFF\}$ **faça**
        se $|F_j^{\diamond} - F_i'| > t - |F_i'|$ **então** $auxF \leftarrow auxF \cup \{j\}$; **fim se**
        se $|FF_j^{\diamond} - F_i'| > t - |F_i'|$ **então** $auxFF \leftarrow auxFF \cup \{j\}$; **fim se**
      fim para

      enquanto $\{auxFF - CheckedSet\} \neq \emptyset$ **faça**
        $u \leftarrow$ selecionar uma unidade de $\{auxFF - CheckedSet\}$;
        $auxFF \leftarrow auxFF \cup FF_u$;
        $auxF \leftarrow auxF \cup F_u$;
        $CheckedSet \leftarrow CheckedSet \cup \{u\}$;
      fim enquanto
    fim se

    se $|auxF| \leq t$ e is_AFS($auxF$) **então**
      retornar $auxF$;
    fim se
  fim para
  ---
fim

Figura 3.2: O código do algoritmo de diagnóstico proposto.

Para completar esta tarefa, esta primeira fase verifica cada uma das unidades i que não possuem nenhuma igualdade nos seus respectivos resultados de comparações, ou seja, o conjunto P_i correspondente é diferente de vazio. Lembre-se que pela Definição 12 se $\nexists r((j,k)_i) = 0$ então $P_i = N(i)$, caso contrário $P_i = \emptyset$. Por esta definição, é importante notar que se $P_i \neq \emptyset$ então $P_i = N(i)$, e portanto, como o grau de cada unidade do sistema é maior ou igual a t, $|P_i| \geq t$. Além disso, se $|P_i| \geq t+2$ então a unidade i está falha, pois o número de unidades falhas é no máximo t e então ao menos uma das comparações realizadas por i deveria resultar em igualdade caso i fosse sem-falha. Por enquanto, nenhuma conclusão pode ser tirada sobre as unidades i tal que $|P_i| = t$ ou $|P_i| = t+1$. Estas unidades são verificadas a seguir. Além disso, é importante destacar que – pelos testes realizados na linha 6 e na linha 12 – esta primeira fase será executada apenas se existir alguma unidade i tal que $|P_i| = t+1$ ou $|P_i| = t$.

Nas linhas 6–10 o algoritmo então verifica cada unidade i tal que $|P_i| = t+1$. Se i está sem-falha então é possível dizer que existe exatamente uma unidade sem-falha $j \in P_i$, pois o número de unidades falhas é menor ou igual a t. O algoritmo então verifica, para cada $j \in P_i$, se $P_i - \{j\}$ é um AFS. Se um AFS for encontrado então este é o conjunto real de unidades falhas F e o algoritmo termina. Caso contrário, se nenhum AFS foi encontrado, pode-se dizer que a unidade i verificada é falha e o algoritmo simplesmente ignora esta unidade e continua normalmente.

Um exemplo da situação na qual $|P_i| = t+1$ é mostrado na Figura 3.3. Neste exemplo, $N = 9$, $t = 4$ e as unidades u_3, u_4, u_8 e u_9 são falhas. Nesta figura, a unidade sem-falha u_1 é vizinha de todas as unidades falhas e também da unidade sem-falha u_2. Note que apesar da unidade u_1 ter uma unidade sem-falha como vizinha, nenhuma de suas comparações resultará em igualdade e portanto $|P_{u_1}| = t+1 = 5$.

Uma estratégia similar é empregada nas linhas 12–27, nas quais o algoritmo verifica cada unidade i tal que $|P_i| = t$. Neste caso se i é uma unidade sem-falha então existe no máximo uma unidade sem-falha $j \in P_i$ e então uma das seguintes três situações pode ocorrer:

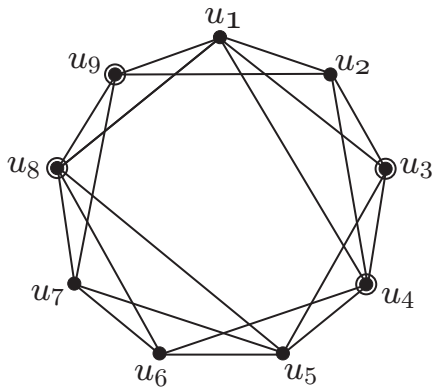


Figura 3.3: Exemplo de um sistema com 9 unidade, $t = 4$. As unidades u_3, u_4, u_8 e u_9 são falhas. A unidade u_1 é vizinha de todas as unidades falhas e também da unidade u_2.

(a.1) P_i pode ser um AFS; esta situação é verificada na linha 13.

(a.2) Para cada $j \in P_i$, $P_i - \{j\}$ (um conjunto com $t-1$ unidades) pode ser um AFS; esta situação é verificada nas linhas 14–16.

(a.3) Um conjunto $P_i - \{j\} \cup \{x\}$ (conjunto com t unidades) pode ser um AFS, onde $j \in P_i$ e $x \in V - P_i$; esta situação é verificada nas linhas 18–24.

É importante destacar na situação (a.3) (linhas 18–24) que se um conjunto $P_i - \{j\}$ com $t-1$ unidades *não* for um AFS mas existir um conjunto $P_i - \{j\} \cup \{x\}$ com t unidades que *é* um AFS, esta unidade extra x deve pertencer ao conjunto de unidades suspeitas $\mathcal{S}$ correspondente. Pela Definição 16 um conjunto suspeito $\mathcal{S}$ consiste de três unidades. Este conjunto suspeito $\mathcal{S}$ é retornado pela função is_AFS quando ela verifica se o conjunto correspondente $P_i - \{j\}$ é um AFS. Na situação (a.3), como o algoritmo verifica se $P_i - \{j\} \cup \{x\}$ é um AFS, as unidades x podem ser selecionadas deste conjunto de três unidades suspeitas, ao invés de verificar cada unidade $x \in V - P_i$.

Se na execução de um dos passos (a.1), (a.2) ou (a.3) um AFS for encontrado, então aquele AFS é o conjunto de unidades falhas e o algoritmo termina. Caso contrário, se nenhum AFS foi encontrado, então a unidade i é falha e o algoritmo continua normalmente.

Um exemplo desta situação onde $|P_i| = t$ é mostrado na Figura 3.4. Neste exemplo $N = 9$, $t = 4$ e as unidades falhas são as unidades u_3, u_4, u_8 e u_9. Na figura, a unidade sem-falha u_1 é vizinha das unidades falhas u_3, u_8 e u_9, e também é vizinha da unidade sem-falha

u_2. Note que como u_1 é sem-falha, nenhuma de suas comparações resultará em igualdade, $|P_{u_1}| = t = 4$ e o AFS neste caso é o conjunto $P_{u_1} - \{j\} \cup \{x\} = \{u_2, u_3, u_8, u_9\} - \{u_2\} \cup \{u_4\}$.

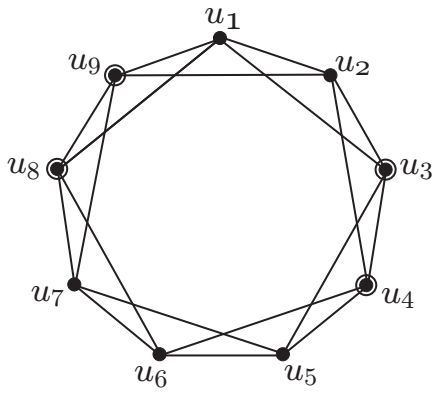


Figura 3.4: Exemplo de um sistema com 9 unidades, $t = 4$. As unidades u_3, u_4, u_8 e u_9 são falhas. A unidade u_1 é vizinha de três das quatro unidades falhas.

Fase 2. Se a segunda fase do algoritmo for alcançada então pode-se dizer que a primeira faze não foi executada ou então que ela não encontrou nenhum AFS. Além disso, se a segunda fase do algoritmo for alcançada, para cada unidade sem-falha i, $|FF_i| \geq 2$, isto é, existe pelo menos uma comparação realizada por i que resulta em igualdade. Em outras palavras, para cada unidade sem-falha i, o conjunto $|FF_i'| > 1$, e além disso, o conjunto F_i' é diferente de vazio desde que exista ao menos uma unidade falha no sistema.

A fase 2 é baseada em uma propriedade particular do sistema: existe ao menos uma unidade sem-falha $i \in V$ tal que F_i' é máximo. Em outras palavras, existem caminhos, que consistem apenas de unidades sem-falha com exceção da unidade final do caminho, que vão da unidade i para o maior conjunto possível de unidades falhas (isto é, para F_i'). Como o *loop* da linha 30 verifica cada unidade do sistema, em algum momento uma unidade i que possui F_i' máximo é verificada.

O primeiro passo dentro do *loop* (linha 31) é inicializar como vazio três conjuntos: *auxFF*, *auxF* e *CheckedSet*. Note que estes conjuntos são temporários e eles são reinicializados em toda iteração deste *loop*. Considerando que a unidade i de uma determinada iteração seja sem-falha, os conjuntos *auxFF* e *auxF* são usados para manter respectiva-

mente conjuntos de unidades sem-falha e de unidades falhas. Por sua vez, $CheckedSet$ é um conjunto usado pelo algoritmo para saber quais unidades u tal que os conjuntos FF_u e F_u correspondentes já foram inseridos respectivamente em $auxFF$ e $auxF$.

Nas linhas 32–35 os conjuntos FF'_i e F'_i são calculados e incluídos em $auxFF$ e $auxF$ respectivamente. É importante lembrar, a partir da definição, que se i é uma unidade sem-falha, FF'_i contém toda unidade sem-falha u tal que existe um caminho $P[i,u]$ que consiste apenas de unidades também sem-falha, e F'_i contém todas as unidades falhas que são vizinhas de qualquer unidade em FF'_i. Além disso, existe um caminho a partir de qualquer unidade sem-falha de FF'_i para cada unidade falha em F'_i, caminho este que consiste apenas de unidades sem-falha com exceção da unidade final do caminho. O código apresentado na Figura 3.5 é uma forma de implementar o cálculo dos conjuntos FF'_i e F'_i e é equivalente às linhas 32–35 do algoritmo de diagnóstico, ou seja, pode substituir as linhas 32–35.

$auxFF \leftarrow \{i\} \cup FF_i$;
$auxF \leftarrow F_i$;
$CheckedSet \leftarrow \{i\}$;
faça
    $u \leftarrow$ selecionar uma unidade de $\{auxFF - CheckedSet\}$;
    $CheckedSet \leftarrow CheckedSet \cup \{u\}$;
    $auxFF \leftarrow auxFF \cup FF_u$;
    $auxF \leftarrow auxF \cup F_u$;
até que $\{auxFF - CheckedSet\} = \emptyset$

Figura 3.5: Um código alternativo às linhas 32–35 do algoritmo $Diag$.

Neste ponto do algoritmo (linha 36) duas situações podem existir:

(b.1) O conjunto F'_i pode conter todas as unidades falhas do sistema; em outras palavras, existe um caminho, que consiste apenas de unidades sem-falha com exceção da unidade final, a partir de cada unidade sem-falha em FF'_i (isto é, de cada unidade em $auxFF$) para cada unidade falha do sistema.

(b.2) O conjunto F'_i não possui todas as unidades falhas; em outras palavras, existe ao menos uma unidade falha tal que não existe um caminho, que

consiste apenas de unidades sem-falha com exceção da unidade final, de qualquer unidade em FF'_i para esta unidade falha.

Exemplos destas duas situações (b.1) e (b.2) são mostrados na Figura 3.6. A figura mostra conjuntos de unidades sem-falha (conjuntos $\mathcal{A}$ e $\mathcal{D}$) e de unidades falhas (conjuntos $\mathcal{B}$ e $\mathcal{C}$). A Figura 3.6(a) mostra uma configuração na qual a situação (b.1) ocorre, isto é, existe um caminho de cada unidade sem-falha em $\mathcal{A}$ para toda unidade falha em $\mathcal{B}$, caminhos estes que consistem apenas de unidades sem-falha com exceção da unidade final. Já a Figura 3.6(b) mostra outra configuração na qual a situação (b.2) ocorre, isto é, não existe tal caminho a partir de todas as unidades sem-falha para as unidades falhas. Mais especificamente, não existe um caminho, que consiste apenas de unidades sem-falha com exceção da unidade final, a partir de qualquer unidade sem-falha em $\mathcal{A}$ para qualquer unidade falha em $\mathcal{C}$.

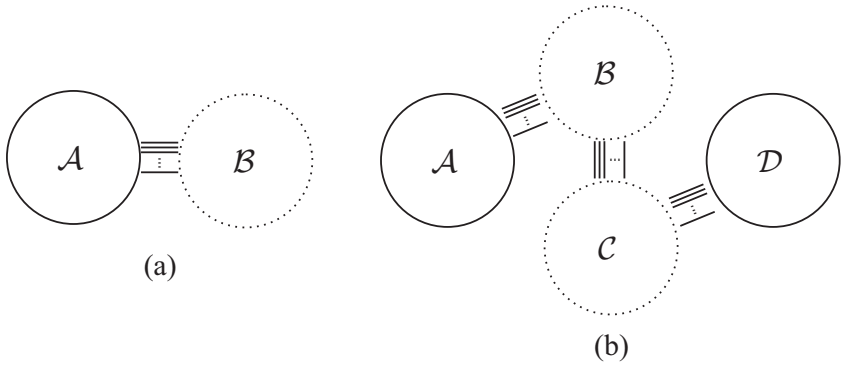


Figura 3.6: Exemplos de configurações que alcançam a fase 2 do algoritmo. As unidades em $\mathcal{A}$ e $\mathcal{D}$ são sem-falha; as unidades em $\mathcal{B}$ e $\mathcal{C}$ são falhas.

As Figuras 3.7 e 3.8 mostram casos particulares das duas configurações de exemplos apresentadas na Figura 3.6. As figuras também mostram conjuntos de ambas unidades falhas e sem-falha. Nos dois exemplos mostrados $t = 4$. As Figuras 3.7 e 3.8 mostram respectivamente sistemas onde a situação (b.1) e (b.2) ocorrem. Na Figura 3.7 $FF'_{u_1} = \{u_1, u_2, u_3, u_4, u_5\}$ e $F'_{u_1} = \{u_6, u_7, u_8, u_9\}$, isto é, todas as unidades falhas estão no conjunto F'_{u_1}. Por outro lado, na Figure 3.8, $FF'_{u_1} = \{u_1, u_2, u_3\}$ e $F'_{u_1} = \{u_4, u_5\}$, isto é, nem toda unidade falha do sistema está no conjunto F'_{u_1} e não existe um caminho, que

consiste apenas de unidades sem-falha com exceção da unidade final, a partir da unidade u_1 para as unidades u_6 e u_7.

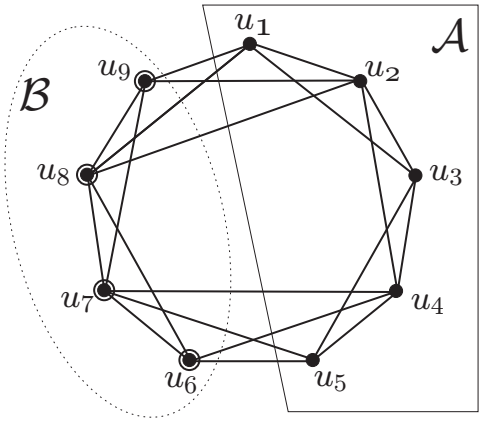


Figura 3.7: Exemplo de um sistema com 9 unidades, $t = 4$, que alcança a fase 2. As unidades em $\mathcal{A}$ são sem-falha; as unidades em $\mathcal{B}$ são falhas.

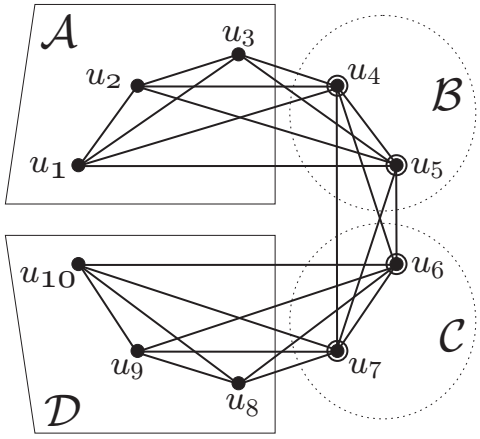


Figura 3.8: Exemplo de um sistema com 10 unidades, $t = 4$, que alcança a fase 2. As unidades em $\mathcal{A}$ e $\mathcal{D}$ são sem-falha; as unidades em $\mathcal{B}$ e $\mathcal{C}$ são falhas.

Quando o algoritmo alcança a linha 36 e a configuração resultante é (b.1), então *auxF* já é o conjunto real de unidades falhas do sistema. Neste caso, as linhas 37–49 não modificam o conjunto *auxF*. Então o algoritmo (nas linhas 51–52), já que *auxF* é um AFS, retorna este conjunto. Por outro lado, se ocorrer a situação (b.2), ainda existem unidades falhas que não estão em F_i' (isto é, ainda não estão em *auxF*), e portanto $|F_i'| < t$.

Também é possível dizer que caso ocorra a configuração (b.2), existe pelo menos mais um conjunto de unidades sem-falha que não estão conectadas a (ou não são vizinhas de)

unidades em FF'_i, isto é, existem unidades sem-falha que ainda não estão em *auxFF*. Além disso, quando i é uma unidade sem-falha que possui conjunto F'_i máximo, ao menos uma das seguintes duas situações sempre ocorre para cada unidade falha $f \notin F'_i$ (este fato é provado no Teorema 5):

(c.1) Existem pelo menos $t - |F'_i| + 1$ unidades testadoras que não estão em F'_i e que testaram f como uma unidade falha, isto é, $|F^{\diamond}_f - F'_i| > t - |F'_i|$;

(c.2) Existe pelo menos um caminho da unidade sem-falha $j \notin FF'_i$ para f, que consiste apenas de unidades sem-falha com exceção da unidade final f *e também* existem pelo menos $t - |F'_i| + 1$ unidades testadoras que não estão em F'_i e que testaram j como sem-falha, isto é, $|FF^{\diamond}_j - F'_i| > t - |F'_i|$.

Para cada unidade f na situação (c.1), como existem no máximo mais $t - |F'_i|$ unidades restantes para serem identificadas como falha, ao menos uma das $t - |F'_i| + 1$ unidades – que testaram f como falha – é sem-falha. Portanto f é uma unidade falha e é incluída no conjunto *auxF* (linha 39).

Cada unidade falha f que não foi incluída em *auxF* por (c.1) é identificada como falha em (c.2) conforme descrito na sequência. A situação (c.2) é implementada nas linhas 40–48 do algoritmo. Primeiramente (na linha 40) como existem no máximo mais $t - |F'_i|$ unidades restantes para serem identificadas como falha, ao menos uma das $t - |F'_i| + 1$ unidades que testaram j como sem-falha é também sem-falha. Então a unidade j é incluída no conjunto *auxFF*. Na sequência (linhas 43–48) as unidades sem-falha alcançáveis a partir de j (através de caminhos que consistem apenas de unidades sem-falha) são incluídas em *auxFF*; e também toda unidade falha f tal que existe pelo menos um caminho, que consiste apenas de unidades sem-falha com exceção da unidade final, partindo de j para f, são também incluídas em *auxF*.

Finalmente (linhas 51–52) se i é uma unidade sem-falha tal que F'_i é máximo, *auxF* é o AFS que corresponde ao conjunto das unidades falhas do sistema (este fato é provado no Teorema 6).

A Figura 3.9 mostra dois exemplos de sistemas que alcançam a segunda fase do algoritmo, nos quais ocorrem, respectivamente, as situações (c.1) e (c.2). O objetivo destas Figuras 3.9(a) e (b) é exemplificar o funcionamento do algoritmo *Diag* para cada uma destas duas situações. Em ambos os exemplos as unidades em $\mathcal{A}$ e $\mathcal{D}$ são sem-falha, e as unidades em $\mathcal{B}$ e $\mathcal{C}$ são falhas. Além disso, $t = 4$.

Considere primeiramente a Figura 3.9(a), caso onde ocorre a situação (c.1). Considere também que o algoritmo está iniciando o *loop* da linha 30, e a unidade i daquela iteração é a unidade u_1. Repare que $FF'_{u_1} = \{u_1, u_2, u_3\}$ e $F'_{u_1} = \{u_4, u_5\}$. Repare também que F'_{u_1} é máximo, pois para qualquer outra unidade sem-falha j, $|F'_j| \leq |F'_{u_1}|$, ou seja, $|F'_j| \leq 2$. Na linha 34, *auxF* é atribuído com as unidades em F'_i. Neste momento, repare que o *loop* da linha 38 será executado para cada uma das unidades u_6, u_7, u_8, u_9 e u_{10}. No caso específico das duas primeiras unidades $j = u_6, u_7$, $|F^{\diamond}_j - F'_{u_1}| > t - |F'_{u_1}|$, pois $F^{\diamond}_j - F'_{u_1}$ possui pelo menos 3 unidades (que são u_8, u_9 e u_{10}), $t = 4$ e $|F'_{u_1}| = 2$, e então $3 > 4 - 2$. Portanto, a linha 39 adiciona em *auxF* cada uma das unidades falhas u_6 e u_7 que ainda não estavam naquele conjunto, identificando todas as unidades falhas do sistema.

Por fim, considere agora a Figura 3.9(b), caso onde ocorre a situação (c.2). Considere que o algoritmo também está iniciando o *loop* da linha 30 e a unidade i daquela iteração é a unidade u_1. Repare novamente que $FF'_{u_1} = \{u_1, u_2, u_3\}$ e $F'_{u_1} = \{u_4, u_5\}$. Repare

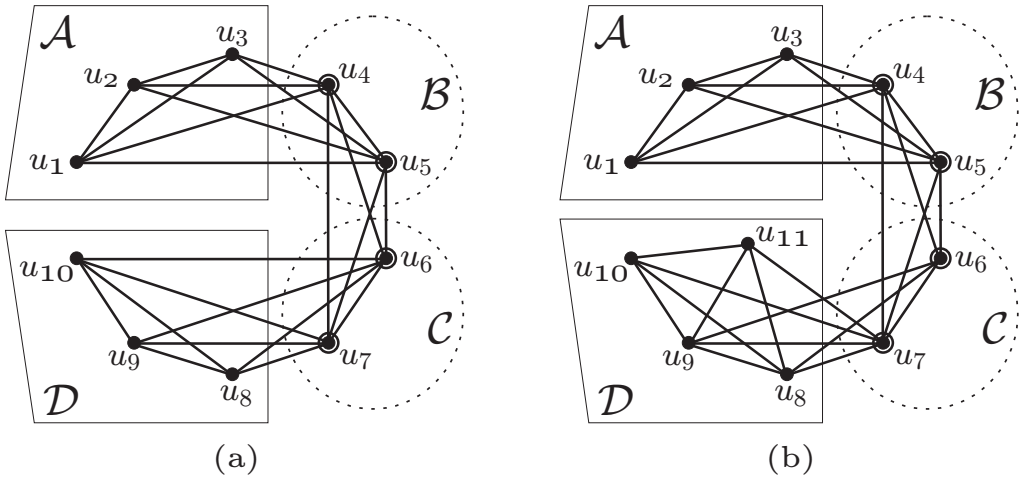


Figura 3.9: Exemplos de um sistemas com $t = 4$ onde ocorre as situações (c.1) e (c.2), respectivamente. As unidades em $\mathcal{A}$ e $\mathcal{D}$ são sem-falha; as unidades em $\mathcal{B}$ e $\mathcal{C}$ são falhas.

também que F'_{u_1} é máximo. Na linha 34, *auxF* é atribuído com as unidades em F'_i, isto é, $auxF = \{u_4, u_5\}$. Neste momento, o *loop* da linha 38 será executado para cada uma das unidades u_6, u_7, u_8, u_9, u_{10} e u_{11}. Para o caso específico da unidade u_7, $|F^{\diamond}_{u_7} - F'_{u_1}| > t - |F'_{u_1}|$, pois $F^{\diamond}_{u_7} - F'_{u_1}$ possui pelo menos 3 unidades, $t = 4$ e $|F'_{u_1}| = 2$, e então $3 > 4 - 2$. Portanto, a linha 39 adiciona a unidade u_7 em *auxF*.

Por outro lado, repare que neste exemplo a unidade u_6 ainda não foi identificada como falha e ainda não está em *auxF*. Isso se deve ao fato de que, neste exemplo – quando comparado a exemplo da Figura 3.9(a) – a aresta (u_6, u_{10}) foi removida. Entretanto, existe uma nova unidade sem-falha u_{11}, que está conectada à u_{10}, para que o grau de u_{10} continue maior ou igual a t (lembre que neste exemplo $t = 4$). Portanto, para esta unidade u_{10}, agora $|FF^{\diamond}_{u_{10}} - F'_{u_1}| > t - |F'_{u_1}|$, pois $FF^{\diamond}_{u_{10}} - F'_{u_1}$ possui pelo menos 3 unidades (que são u_8, u_9 e u_{11}), e então $3 > 4 - 2$. Portanto, a linha 40 atribui em *auxFF* a unidade u_{10} e na sequência, o *loop* das linhas 43–48 insere as demais unidade sem-falha u_8, u_9 e u_{11} em *auxFF*. Por fim, a linha 46 insere a última unidade falha u_6 em *auxF*, pois $u_6 \in F_{u_8}$, identificando assim todas as unidades falhas do sistema.

3.3 Provas de Correção e Análise de Complexidade

Esta seção apresenta as provas de correção da função is_AFS e do algoritmo $Diag$. A seção está organizada e apresenta os lemas, teoremas e corolários na seguinte ordem: da função is_AFS, da primeira fase do algoritmo de diagnóstico proposto, e da segunda fase do algoritmo de diagnostico. Por fim, esta seção ainda apresenta a análise da complexidade do algoritmo de diagnóstico proposto.

Provas de Correção da Função is_AFS

Lema 1: *Para qualquer unidade i tal que $P_i = \emptyset$, $|FF_i| \geq 2$.*

Prova: Por contradição suponha que $P_i = \emptyset$ e que $|FF_i| \leq 1$. Pelas Definições 6 e 12 respectivamente, se $r((j,k)_i) = 0$ então $j, k \in FF_i$, e se $\nexists r((j,k)_i) = 0$ onde

$j, k \in N(i)$ então $P_i = N(i)$, caso contrário $P_i = \emptyset$. Se $P_i = \emptyset$ então existe pelo menos uma comparação $r((j,k)_i) = 0$. Portanto, pelo menos as duas unidades $j, k \in FF_i$. □

Lema 2: *Para qualquer unidade i tal que $P_i \neq \emptyset$, $FF_i = \emptyset$ e $F_i = \emptyset$.*

Prova: Por contradição suponha que $P_i \neq \emptyset$ mas (a) $FF_i \neq \emptyset$ ou (b) $F_i \neq \emptyset$. Considere o caso (a): se $FF_i \neq \emptyset$ então pela Definição 12 $P_i = \emptyset$. Agora considere o caso (b): se $F_i \neq \emptyset$ então pela Definição 7 existe ao menos uma comparação $(j,k)_i$ onde $r((j,k)_i) = 1$ e $k \in FF_i$. Como pelo menos $k \in FF_i$, então $FF_i \neq \emptyset$. Portanto novamente pela Definição 12 $P_i = \emptyset$. □

Teorema 1: *A função is_AFS determina corretamente quando um conjunto de entrada $U \subset V$ é um AFS, isto é, se para quaisquer três unidades i, j, k tal que $(j,k)_i \in C$, $j, k \in N(i)$ e $j \neq k$, as seguintes duas condições são satisfeitas:*

(a) se $i \in V - U$ e $j, k \in V - U$ então $r((j,k)_i) = 0$

(b) se $i \in V - U$ e $\{j,k\} \cap U \neq \emptyset$ então $r((j,k)_i) = 1$

Prova: Sem perda de generalidade, é possível dizer que um conjunto $U \subset V$ *não* é um AFS, se para quaisquer três unidades i, j, k tal que $(j,k)_i \in C$, $j, k \in N(i)$ e $j \neq k$, uma das seguintes duas condições são satisfeitas:

(i) $i \in V - U$ e $\{j,k\} \cap U \neq \emptyset$ e $r((j,k)_i) = 0$

(ii) $i \in V - U$ e $j, k \in V - U$ e $r((j,k)_i) = 1$

Na linha 2 da função *is_AFS* (Figura 3.1), somente unidades $i \in V - U$ são consideradas. Por definição, os conjuntos FF_i, F_i e P_i são conjuntos compostos por unidades que são comparadas pela unidade testadora i.

Pelos Lemas 1 e 2, para cada unidade i, se $P_i \neq \emptyset$ então $FF_i = \emptyset$ e $F_i = \emptyset$, caso contrário, ao menos $FF_i \neq \emptyset$. Primeiramente considere o caso no qual $P_i = \emptyset$, isto é, pelo menos $FF_i \neq \emptyset$.

Primeiro caso: $P_i = \emptyset$.

As linhas 4, 5 e 7 da função is_AFS verificam se qualquer unidade $j \in FF_i$ está também no conjunto U. Se existir tal situação, a função retorna "false", isto é, indica que o conjunto U não é um AFS. Pela Definição 6, se $j \in FF_i$ então existe uma comparação $r((j,k)_i) = 0$. Neste caso onde $r((j,k)_i) = 0$, se $i \in V - U$ e $j \in U$ então $\{j,k\} \cap U \neq \emptyset$. Esta é exatamente a situação da condição (i).

As linhas 11, 12 e 14 verificam se qualquer unidade $j \in F_i$ está também no conjunto $V - U$. Se existir tal situação, a função retorna "false". Pela Definição 7, se $j \in F_i$ existe uma comparação $(j,k)_i$ onde $r((j,k)_i) = 1$ e $k \in FF_i$. Como $k \in FF_i$ existe também uma comparação $r((k,x)_i) = 0$ e então $k \in V - U$. Portanto, estas linhas verificam se $i \in V - U$ e $j,k \in V - U$ e $r((j,k)_i) = 1$, isto é, U não pode ser um AFS. Esta é exatamente a situação da condição (ii).

Como no primeiro caso $P_i = \emptyset$, as linhas 19–30 não possuem nenhum efeito sobre o resultado de retorno da função.

Segundo caso: $P_i \neq \emptyset$.

Como $P_i \neq \emptyset$ então $FF_i = \emptyset$ e $F_i = \emptyset$. Além disso, como $P_i \neq \emptyset$ pela Definição 12 não existe nenhuma comparação $(j,k)_i \mid r((j,k)_i) = 0$, isto é, todas as comparações realizadas por i resultaram em diferença.

Como $FF_i = \emptyset$ e $F_i = \emptyset$ então o código das linhas 4–16 não possui nenhum efeito sobre o resultado de retorno da função. Por outro lado, as linhas 19–21 verificam quantas unidades de P_i estão também em $V - U$. Considere que existem pelo menos as seguintes duas unidades em tal situação: unidades a e b. Então $a,b \in P_i$ e $a,b \in V - U$. Como $a,b \in P_i$, e toda comparação realizada por i resultou em diferença, então $r((a,b)_i) = 1$. Se existir tal situação (linhas 27–28) a função retorna "false". Esta é exatamente a situação testada na condição (ii). Além disso, como $\nexists (j,k)_i \mid r((j,k)_i) = 0$, a condição (i) não se aplica e a função is_AFS não verifica esta condição quando $P_i \neq \emptyset$. □

Corolário 1: *As linhas 22–23 da função* is_AFS *retornam no array* $\mathcal{S}$[1,2,3] *três unidades* i,j,k *de uma comparação* $(j,k)_i$ *tal que* $i,j,k \in V - U$ *e* $r((j,k)_i) = 1$.

Prova: Nas linhas 21–22 o *array* $\mathcal{S}$[1,2,3] é atribuído com a unidade testadora i e duas unidades $a, b \in P_i$. Por definição, se $P_i \neq \emptyset$, $P_i = N(i)$ e $\nexists r((j,k)_i) = 0 \mid j,k \in N(i)$. Como no modelo MM* todos os vizinhos de cada unidade i são comparadas em pares, $(a,b)_i$ é uma das comparações realizadas por i e então $r((a,b)_i) = 1$. Além disso, pela linha 20 $a, b \in V - U$. □

Provas de Correção da Fase 1 do Algoritmo *Diag*

Lema 3: *Se existe uma unidade i tal que $|P_i| > t+1$, então i é uma unidade falha.*

Prova: Por definição, se $|P_i| > 0$ então $P_i = N(i)$ e todas as comparações realizadas por i resultaram em diferença. Neste caso $|N(i)| > t+1$. Por contradição, suponha que existe uma unidade sem-falha i tal que $|N(i)| > t+1$ e todas as comparações realizadas por i indicam diferença: isto é, $r((j,k)_i) = 1$. Como o número de unidades falhas é no máximo t e $|N(i)| > t+1$, pelo menos duas unidades de $N(i)$ são sem-falha; como a unidade comparadora i é também sem-falha, a comparação das duas unidades sem-falha deve resultar em igualdade, o que é uma contradição. □

Lema 4: *Se existe uma unidade i tal que $|P_i| = t+1$ então ou a unidade i é falha, ou então $P_i - \{j\}$ é um AFS, $j \in P_i$.*

Prova: Por definição, se $|P_i| > 0$ então $P_i = N(i)$ e todas as comparações realizadas por i resultaram em diferença. Neste caso $|N(i)| = t+1$.

Se existirem duas unidades $j, k \in N(i)$ que são sem-falha, então i é uma unidade falha, caso contrário a comparação $(j,k)_i$ deveria resultar em igualdade. Como o número máximo de unidades falhas no sistema é t, se a unidade i é sem-falha deve existir uma unidade sem-falha $j \in N(i)$. Como existe apenas um t-AFS (um conjunto de unidades que é um AFS e que possui no máximo t unidades), então $P_i - \{j\}$ deve ser o AFS. □

Lema 5: *Se existir uma unidade i tal que $|P_i| = t$, então ou (i) a unidade i é falha, ou (ii) P_i é um AFS, ou (iii) $|F \cap P_i| = t-1$.*

Prova: Por definição, se $|P_i| > 0$ então $P_i = N(i)$ e todas as comparações realizadas por i resultaram em diferença. Neste caso $|N(i)| = t$. Se existirem duas unidades sem-falha $j, k \in N(i)$, então a unidade i é falha. Agora considere o caso no qual a unidade i é sem-falha. Como o número máximo de unidades falhas no sistema é t e $|P_i| = t$, então podem existir zero ou uma unidade sem-falha em $N(i)$. Portanto, como existe somente um t-AFS então $t - 1 \leq |F| \leq t$ e $|F \cap P_i| \geq t - 1$. □

Em outras palavras, o Lema 5 prova que, se existir uma unidade i tal que $|P_i| = t$, uma e apenas uma das seguintes quatro condições é verdadeira:

(i) i é uma unidade falha;

(ii) P_i é um AFS;

(iii) um conjunto $P_i - \{j\}$ com $t - 1$ unidades é um AFS, $j \in P_i$;

(iv) um conjunto $P_i - \{j\} \cup \{x\}$ com t unidades é um AFS, $j \in P_i$ e $x \in V - P_i$.

Corolário 2: *Se existir uma unidade sem-falha i tal que $P_i \neq \emptyset$, então ou (i) P_i é um AFS, ou (ii) $P_i - \{j\}$ é um AFS, $j \in P_i$, ou (iii) $P_i - \{j\} \cup \{x\}$ é um AFS, $x \in V - P_i$.*

Prova: Trivial a partir dos Lemas 4 e 5. □

Lema 6: *Não existe nenhuma unidade i tal que $0 < |P_i| < t$.*

Prova: Por definição, $P_i = \{N(i)$ se $\nexists r((j,k)_i) = 0\}$, caso contrário $P_i = \emptyset$. Como $d(i) \geq t$ e portanto $|N(i)| \geq t$, caso $P_i \neq \emptyset$ então $|P_i| = |N(i)|$, isto é, $|P_i| \geq t$. □

Lema 7: *Para qualquer unidade i tal que $|P_i| > 0$, então ou (i) é possível encontrar um conjunto U que é um AFS tal que $t - 1 \leq |U \cap P_i| \leq t$, ou (ii) a unidade i é falha.*

Prova: Trivial a partir dos Lemas 3, 4, 5 e 6:

Pelo Lema 3, toda unidade i tal que $|P_i| > t + 1$ é uma unidade falha.

Pelo Lema 4, toda unidade i tal que $|P_i| = t + 1$ é uma unidade falha ou então existe um conjunto U que é um AFS e $|P_i \cap U| = t$.

Pelo Lema 5, toda unidade i tal que $|P_i| = t$ é falha ou então existe um conjunto U que é um AFS e $t - 1 \leq |P_i \cap U| \leq t$.

Finalmente, pelo Lema 6, não existe nenhuma unidade i tal que $0 < |P_i| < t$. □

Lema 8: *Considere uma unidade i tal que $|P_i| = t$, P_i não é um AFS e $\forall j \in P_i, P_i - \{j\}$ não é um AFS. Se existir uma unidade x tal que $P_i - \{j\} \cup \{x\}$ é um AFS, então x pertence ao conjunto de unidades suspeitas $\mathcal{S}$ que é retornado quando se verifica se o conjunto correspondente $P_i - \{j\}$ é um AFS.*

Prova: Pela Definição 16, um conjunto de unidades suspeitas $\mathcal{S}$ consiste de três unidades $\{s_1, s_2, s_3\}$ tal que a comparação $(s_2, s_3)_{s_1} \in C$ não é válida para uma das seguintes duas condições, quando se verifica se um conjunto $U = P_i - \{j\}$ é um AFS:

(a) se $s_1 \in V - U$ e $s_2, s_3 \in V - U$ então $r((s_2, s_3)_{s_1}) = 0$

(b) se $s_1 \in V - U$ e $\{s_2, s_3\} \cap U \neq \emptyset$ então $r((s_2, s_3)_{s_1}) = 1$

Isto significa que para o conjunto $U = P_i - \{j\}$ se a condição (a) é a condição que não é válida, $s_1 \in V - U$ e s_2 ou s_3 (ou ambas as unidades s_2 e s_3) estão em U. Entretanto, e a condição (b) é a condição que não é válida, $\{s_1, s_2, s_3\} \notin U$.

Como existe somente um único t-AFS no sistema, e este lema apresenta que $P_i - \{j\}$ não é um AFS mas existe uma única unidade restante x tal que $P_i - \{j\} \cup \{x\}$ é um AFS, isto implica que considerando o conjunto $U = P_i - \{j\} \cup \{x\}$ as duas condições do AFS devem ser válidas para a comparação $(s_2, s_3)_{s_1}$. Este fato permite apenas uma de duas possibilidades:

— se a condição (a) não é a condição válida quando se verifica se o conjunto $U = P_i - \{j\}$ é um AFS, então s_1 deve ser a unidade falha;

— caso a condição (b) não é a condição válida, então apenas uma das unidades s_1, s_2 ou s_3 deve ser a unidade falha restante.

Portanto $x \in \{s_1, s_2, s_3\} = \mathcal{S}$. □

A partir dos oito lemas acima, é possível concluir que a fase 2 do algoritmo é executada se e somente se não existe uma unidade i no sistema tal que: (a) todas as comparações realizadas por i retornaram diferença, e (b) a unidade i é vizinha de t ou $t-1$ unidades falhas; caso contrário, um AFS é encontrado na fase 1.

Teorema 2: *Se a fase 1 do algoritmo não encontra um conjunto U que é um AFS então todas as unidades sem-falha do sistema possuem pelo menos duas outras unidades sem-falha como vizinhas.*

Prova: Pelo Lema 3 não existe unidade sem-falha i tal que $|P_i| > t+1$.

Pelo Lema 6 não existe nenhuma unidade i tal que $0 < |P_i| < t$.

Pelo Lema 4 não existe unidade sem-falha i tal que $|P_i| = t+1$ e $P_i - \{j\}$ não é um AFS, $j \in P_i$.

Pelo Lema 5 e pelo Corolário 1 não existe unidade sem-falha i tal que $|P_i| = t$ e também, (i) P_i seja um AFS, ou (ii) um conjunto $P_i - \{j\}, j \in P_i$ (com $t-1$ unidades) seja um AFS, ou (iii) um conjunto $P_i - \{j\} \cup \{x\}, j \in P_i$ e $x \in V - P_i$ (com t unidades) seja um AFS. □

Provas de Correção da Fase 2 do Algoritmo *Diag*

Teorema 3: *Se em um sistema t-diagnosticável existe uma unidade sem-falha i a partir da qual existe pelo menos um caminho $P[i,x]$, que consiste apenas de unidades sem-falha com exceção da unidade final do caminho, para cada uma das unidades falhas x, então F_i' é um AFS.*

Prova: Pela Definição 14 quando i é sem-falha então o conjunto F_i' contém todas as unidades falhas tal que existe um caminho a partir de i para cada unidade falha em F_i', que consiste apenas de unidades sem-falha com exceção da unidade final.

Como em um sistema t-diagnosticável existe somente um AFS com até t unidades, se todas as unidades falhas do sistema estiverem em F_i', então F_i' é um AFS; caso contrário existiria ao menos uma unidade falha $f \notin F_i'$ de forma que não existiria um caminho que consistem apenas de unidades sem-falha com exceção da unidade final, a partir de i para f. □

Teorema 4: *Considere um sistema com N unidades tal que no máximo t são falhas. Se i é uma unidade sem-falha, $|F_i'| < t$, e existe uma unidade $j \in V - F_i'$ tal que $|F_j^\diamond - F_i'| \geq t - |F_i'| + 1$, então j é uma unidade falha.*

Prova: Por definição, se $a \in F_b$ então $b \in F_a^\diamond$. Quando i é sem-falha então pela Definição 14 toda unidade de F_i' é falha. Como $|F_i'| < t$ então devem existir no máximo mais $t - |F'|$ unidades falhas em $V - F_i'$. Como $|F_j^\diamond - F_i'| \geq t - |F_i'| + 1$, então pelo menos uma unidade $x \in F_j^\diamond - F_i'$ é uma unidade sem-falha. Como x é uma unidade sem-falha e $x \in F_j^\diamond$ então $j \in F_x$. Portanto j é uma unidade falha. □

Lema 9: *Considere um sistema com N unidades tal que no máximo t são falhas. Se i é uma unidade sem-falha, $|F_i'| < t$, e existe uma unidade $j \in V - F_i'$ tal que $|FF_j^\diamond - F_i'| \geq t - |F_i'| + 1$, então j é uma unidade sem-falha.*

Prova: A prova deste teorema é análoga à prova do Teorema 4. Como i é uma unidade sem-falha então toda unidade de F_i' é falha. Como $|F_i'| < t$ então existe no máximo mais $t - |F'|$ unidades falhas em $V - F_i'$. Como $|FF_j^\diamond - F_i'| \geq t - |F_i'| + 1$, então pelo menos uma unidade $x \in FF_j^\diamond - F_i'$ é sem-falha. Como x é sem-falha e $x \in FF_j^\diamond$ então $j \in FF_x$. Portanto j é uma unidade sem-falha. □

Teorema 5: *Considere um sistema t-diagnosticável tal que:*

(a) $N \geq 2t + 1$;

(b) para cada $i \in V$, $|N(i)| \geq t$;

(c) para cara cada unidade sem-falha i, $|FF_i| \geq 2$;

(d) $|\xi(\overline{G_F})| \geq 2$*;*

(e) para cada unidade sem-falha i*,* F'_i *não é um AFS;*

(f) existe uma unidade sem-falha $k \in V$ *tal que* F'_k *é máximo;*

(g) $G_k = (V_k, E_k)$ *é o componente de* $\xi(\overline{G_F})$ *tal que* $k \in V_k$*.*

Para cada componente $G_x = (V_x, E_x)$ *tal que* $G_x \in \xi(\overline{G_F})$*,* $G_x \neq G_k$ *e* x *é uma unidade sem-falha em* V_x*, ao menos uma das seguintes condições são satisfeitas:*

(i) Cada unidade falha $y \in F'_x$ *possui pelo menos* $t-|F'_k|+1$ *unidades vizinhas sem-falha, isto é,* $|F^{\diamond}_y - F'_k| \geq t - |F'_k| + 1$*;*

(ii) Pelo menos uma unidade sem-falha $w \in V_x$ *possui ao menos* $t - |F'_k| + 1$ *unidades vizinhas sem-falha, isto é,* $|FF^{\diamond}_w - F'_k| \geq t - |F'_k| + 1$*.*

Prova: Como para cada unidade $v \in V_x$, $|N(v)| \geq t$, então $|V_x| + |F'_x| \geq t + 1$.

Considere o caso onde $|V_x| + |F'_x| = t + 1$. Neste caso, para cada unidade $x \in V_x$, $|N(x)| = t$, e portanto toda unidade $x \in V_x$ deve estar conectada (ser vizinha) a cada uma das outras unidades em V_x e também a cada uma das unidades em F'_x. Como $|V_x| + |F'_x| = t + 1$, então $|V_x| = t + 1 - |F'_x|$, e então para cada unidade $y \in F'_x$, $|F^{\diamond}_y| \geq t+1-|F'_x|$. Como o conjunto F'_k é máximo, $|F'_x| \leq |F'_k|$, e então $|F^{\diamond}_y| \geq t+1-|F'_k|$. Como cada unidade em V_x é sem-falha, então $V_x \cap F'_k = \emptyset$, e portanto $|F^{\diamond}_y - F'_k| \geq t+1-|F'_k|$, o que satisfaz a condição (i).

Neste primeiro caso $|V_x| + |F'_x| = t + 1$ e então $|V_x| = t + 1 - |F'_x|$, e como cada unidade $w \in V_x$ é vizinha de (conectada a) cada uma das outras unidades em V_x, então $|N(w) \cap V_x| = t - |F'_x|$. Este fato é utilizado no próximo caso.

Agora considere o segundo caso onde $|V_x| + |F'_x| > t + 1$. Este caso é equivalente ao anterior (onde $|V_x| + |F'_x| = t + 1$) com a adição de pelo menos uma nova unidade $z \in V_x$. Como $|N(z)|$ deve ser maior ou igual a t, z é conectada a pelo menos $t - |F'_x|$ unidades de V_x. Considere, por exemplo, que w é uma das unidades de V_x conectadas a

z. Então $|N(w) \cap V_x| \geq t - |F'_x| + 1$. Como o conjunto F'_k é máximo, $|F'_x| \leq |F'_k|$, então $|N(w) \cap V_x| \geq t - |F'_k| + 1$. Como em $N(w) \cap V_x$ existem somente unidades sem-falha, então $|FF^{\diamond}_w - F'_k| \geq t - |F'_k| + 1$, o que satisfaz a condição (ii). □

Teorema 6: *Considere um sistema t-diagnosticável com N unidades, onde: (a) $N \geq 2t + 1$; (b) para cada unidade $i \in V$, $|N(i)| \geq t$; e (c) para cada unidade sem-falha i, $|FF_i| \geq 2$.*

A fase 2 do algoritmo identifica todas as unidades falhas desde que cada unidade sem-falha possua pelo menos duas unidades vizinhas sem-falha.

Prova: De acordo com o Teorema 2 se a fase 2 do algoritmo for alcançada, então todas as unidades sem-falha do sistema possuem ao menos duas unidades vizinhas sem-falha.

De acordo com o Teorema 3 se existe uma unidade sem-falha i em um sistema t-diagnosticável tal que cada unidade falha x está em F'_i, então F'_i é um AFS.

Por fim, o Teorema 5 considera os casos onde $|\xi(\overline{G_F})| \geq 2$, $G_k = (V_k, E_k)$ é um componente de $\xi(\overline{G_F})$, $k \in V_k$ e o conjunto F'_k é máximo. De acordo com o teorema, para cada outro componente $G_x = (V_x, E_x)$ tal que $G_x \neq G_k$ e x é uma unidade sem-falha de V_x, uma ou ambas as seguintes condições são satisfeitas:

(i) Cada unidade falha $y \in F'_x$ possui pelo menos $t-|F'_k|+1$ unidades vizinhas sem-falha, isto é, $|F^{\diamond}_y - F'_k| \geq t - |F'_k| + 1$;

(ii) Pelo menos uma unidade sem-falha $w \in V_x$ possui ao menos $t - |F'_k| + 1$ unidades vizinhas sem-falha, isto é, $|FF^{\diamond}_w - F'_k| \geq t - |F'_k| + 1$.

Se a condição (i) for satisfeita para um dado componente G_x então a fase 2 do algoritmo classifica todas as unidades F'_x como falhas (linhas 39 do algoritmo). Por outro lado, se a condição (ii) for satisfeita pelo menos uma unidade $w \in V_x$ é corretamente classificada como sem-falha (linha 40 do algoritmo). Com base nesta unidade sem-falha w, o código das linhas 42–47 do algoritmo classifica todas as unidades falhas em $F'_w = F'_x$ como falhas. □

Análise de Complexidade

Para analisar a complexidade do algoritmo, primeiramente é importante notar que o tamanho da síndrome de comparações σ é $O(\Delta^2 N)$. O cálculo dos conjuntos FF_i, F_i, P_i, $FF_i^\diamond$ e $F_i^\diamond$ envolve uma verificação de cada um dos elementos da síndrome, sendo portanto $O(\Delta^2 N)$.

A função is_AFS (Figura 3.1) possui ordem de complexidade $O(\Delta N)$. O *loop* mais externo (linha 2) é executado no máximo N vezes. Cada um dos três *loops* internos nas linhas 4, 11 e 19 (que não estão aninhados) são executados no máximo Δ vezes, pois $|FF_i| \leq \Delta$, $|F_i| \leq \Delta$ e $|P_i| \leq \Delta$. Note que os conjuntos $CompFF_{i,j}$ e $CompF_{i,j}$ (linhas 6 e 13) podem ser calculados em $O(1)$ pois as três unidades de cada um destes três conjuntos podem ser armazenadas juntamente com cada unidade dos conjuntos FF_i e F_i correspondentes, quando estes conjuntos são calculados.

A complexidade do algoritmo de diagnóstico apresentado na Figura 3.2 é $O(\Delta N^2)$ se $t^2 < N$ ou $O(t^2 \Delta N)$ no caso contrário – onde a complexidade da fase 1 é $O(t^2 \Delta N)$ e a complexidade da fase 2 é $O(\Delta N^2)$, como mostrado na sequência.

Primeiramente, considere a fase 1. O *loop* mais externo (na linha 6) é executado no máximo $t + 1$ vezes, pois se a função is_AFS na linha 8 for executada por uma unidade sem-falha, faz com que o algoritmo termine e retorne o conjunto de unidades falhas (Lema 4). O *loop* interno na linha 7 também é executado no máximo $t + 1$ vezes, pois o *loop* externo avalia apenas as unidades onde $|P_i| = t + 1$. Além disso, a função is_AFS é $O(\Delta N)$. A ordem de complexidade do código nas linhas 12–27 é exatamente a mesma ordem de complexidade do código nas linhas 6–10, isto é, $O(t\Delta^2 N)$: ambos os *loops* nas linhas 12 e 14 executam no máximo t vezes e a função is_AFS é $O(\Delta N)$.

A complexidade da fase 2 é $O(\Delta N^2)$. O *loop* externo na linha 30 é executado no máximo N vezes. Como mostrado na Figura 3.5 as linhas 32–35 (incluindo o cálculo dos conjuntos FF_i' e F_i') pode ser implementada em $O(\Delta N)$. O código nas linhas 37–41 é $O(tN)$: o *loop* na linha 38 é executado no máximo N vezes; e, devido à condição na linha 37 (isto é, $|F_i'| < t$) o código nas linhas 39 e 40 são $O(t)$. O código nas linhas 43–48 são

$O(\Delta N)$: o *loop* na linha 43 também é executado no máximo N vezes; as operações de conjunto nas linhas 45–46 são, cada uma, executadas em $O(\Delta)$ no pior caso; e as linhas 44 e 47 são $O(1)$. Finalmente, as linhas 51–52 são também $O(\Delta N)$.

Como a síndrome de comparações possui tamanho $O(\Delta^2 N)$, a complexidade do algoritmo proposto é muito próxima da complexidade de percorrer os elementos da síndrome uma única vez.

Considerando um sistema completamente conectado (para o qual $\Delta = \delta = N - 1$) a complexidade do algoritmo proposto é $O(N^3)$ se $t^2 < N$, e $O(t^2 N^2)$ no caso contrário. Para esta mesma topologia a complexidade de ambos o algoritmo de Sengupta e Dahbura é $O(N^5)$ e a complexidade do algoritmo de Yang e Tang também é $O(N^5)$.

3.4 Resultados Experimentais

Esta seção apresenta resultados experimentais obtidos através de simulações do algoritmo de diagnóstico proposto, em sistemas de 9, 16, 64 e 128 unidades. Um total de 10.000 simulações foram executadas: 2,5 mil simulações para cada um dos tamanhos de sistemas simulados. O principal propósito dos experimentos foi verificar o comportamento médio esperado para o algoritmo de diagnóstico para sistemas de topologias arbitrárias.

Os sistemas simulados foram arbitrariamente gerados: para cada simulação o grau mínimo δ das unidades foi aleatoriamente selecionado (uniformemente distribuído) – é importante lembrar que pelas condições de diagnosticabilidades (apresentadas na Seção 2.3 e também resumidas no início deste capítulo) que δ também limita o valor máximo de t; e a quantidade de unidades falhas também foi uniformemente distribuída e foi um número entre 1 e t, isto é, todos os experimentos tiveram pelo menos uma unidade falha. Além disso, o resultado das comparações realizadas por unidades falhas também foi aleatoriamente escolhido. As simulações foram executadas em um computador com processador AMD Phenom 9500 quad-core x64 e 4GB de RAM, executando Linux 64-*bits*, *kernel* versão 2.6.18-238.el5.

A Figura 3.10 apresenta a média do número de testes executados pelo algoritmo de diagnóstico, considerando os quatro diferentes tamanhos de sistemas: 9, 16, 64 e 128 unidades. Além da média do número de testes, as linhas verticais mostram o intervalo de confiança de 95%. É possível notar que a média do número de testes executados pelo algoritmo foi de cerca de $N^{2.5}$ em todos os experimentos. Pode-se também notar através dos intervalos de confiança mostrados que a dispersão do número de testes – que engloba os experimentos com diferentes valores de t variados de forma uniforme – foi pequena.

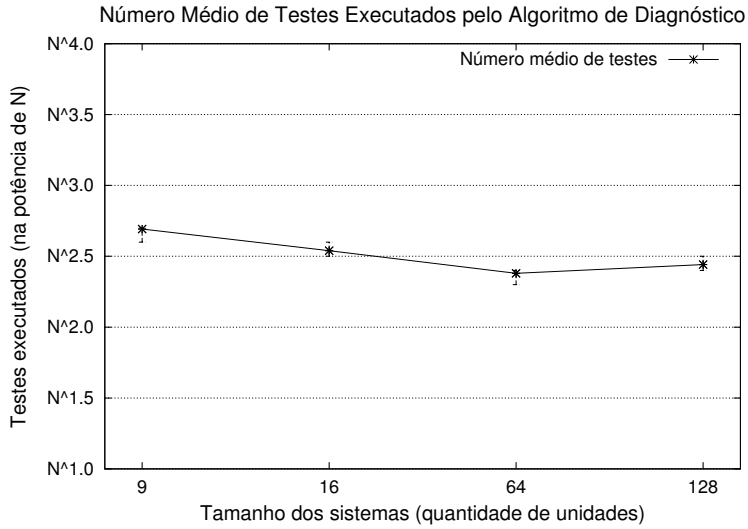


Figura 3.10: Número médio de testes (valor identificado em termos da potência de N) executado pelo algoritmo *Diag*.

O comportamento do algoritmo também foi examinado em termos do número de vezes que verificações de AFS encontraram o conjunto real das unidades falhas do sistema. Em outras palavras, considerando o algoritmo da Figura 3.2 a função is_AFS é chamada sete vezes – foi registrada quantas vezes cada uma destas chamadas encontrou o conjunto real das unidades falhas, fazendo com que o algoritmo terminasse e retornasse o conjunto das unidades falhas encontrado. Pode-se notar que no código do algoritmo existe uma instrução "retornar" nas linhas 8, 13, 16, 19, 21, 23 e 52; as seis primeiras ocorrências estão na fase 1 e somente a última instrução "retornar" na linha 52 está na fase 2. As Figuras 3.11 e 3.12 mostram quantas vezes (em porcentagem) cada um destes pontos "retornar" foi alcançado. A diferença da Figura 3.11 para a Figura 3.12 é que esta última

mostra a porcentagem de forma cumulativa.

Com base na Figura 3.11 pode-se notar que a verificação de AFS da linha 52 foi executada com uma frequência muito maior do que as anteriores. Além disso, com base na Figura 3.12 pode-se perceber que nos experimentos com mais de 16 unidades, todas as verificações de AFS da fase 1 (linhas 8, 13, 16, 19, 21 e 23) encontraram o conjunto de unidades falhas do sistema apenas em menos de 4% das simulações.

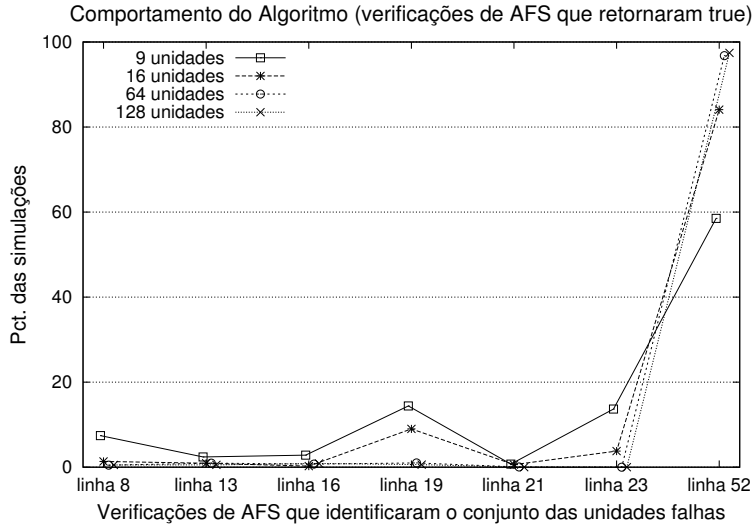


Figura 3.11: Porcentagem dos experimentos nos quais as diferentes verificações de AFS encontram o conjunto das unidades falhas.

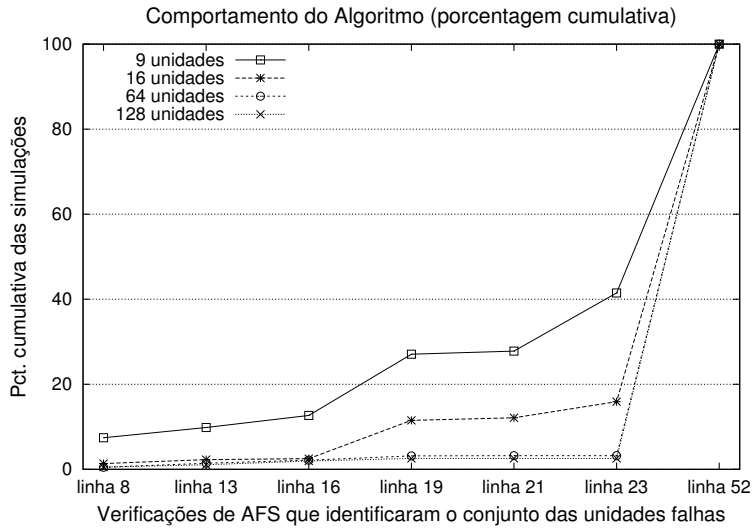


Figura 3.12: Porcentagem cumulativa dos experimentos nos quais as diferentes verificações de AFS encontram o conjunto das unidades falhas.

Como a complexidade da fase 1 é maior do que a complexidade da fase 2, e como a fase 1 é executada somente nos casos onde existe pelo menos uma unidade i tal que $t \leq |P_i| \leq t+1$, também é relevante medir a frequência em que cada uma das fases é executada. É importante lembrar a partir da seção anterior (Seção 3.3) que a complexidade do algoritmo de diagnóstico é $O(\Delta N^2)$ se $t^2 < N$ ou $O(t^2 \Delta N)$ no caso contrário, que são as complexidades da fase 2 e da fase 1, respectivamente.

A Tabela 3.1 mostra a porcentagem dos experimentos que executaram cada uma das fases do algoritmo de diagnóstico. Uma observação é que a soma das duas colunas pode ser maior que 100%, pois é possível que para um determinado experimento ambas as fases foram executadas, isto é, a fase 1 foi executada mas não encontrou nenhum AFS e então a fase 2 foi também executada.

Tamanho dos Sistemas	*Porcentagem das Simulações que* *Executaram a Fase 1*	*Executaram a Fase 2*
9 unidades	42.5%	58.5%
16 unidades	18.9%	84.1%
64 unidades	3.2%	96.8%
128 unidades	2.6%	97.4%

Tabela 3.1: Porcentagem dos experimentos que executaram cada uma das fases do algoritmo de diagnóstico.

Com base na Tabela 3.1 pode-se observar que para os maiores sistemas, apenas uma pequena porcentagem dos experimentos executaram a fase 1. Para ser mais preciso, mais de 96% dos experimentos executaram apenas a fase 2 para sistemas com 64 e 128 unidades. Além disso, com base nestes resultados, pode-se dizer que, para estes maiores sistemas, o algoritmo proposto geralmente não executa a porção de código que possui o pior caso da ordem de complexidade teórica do algoritmo, isto é, em mais de 96% das simulações para estes sistemas, o algoritmo executou apenas a fase 2, que por sua vez é $O(\Delta N^2)$.

CAPÍTULO 4

COMBATE À POLUIÇÃO DE CONTEÚDO EM TRANSMISSÕES AO VIVO EM REDES P2P

Este capítulo apresenta duas novas soluções que utilizam o diagnóstico baseado em comparações para a detecção e combate à poluição de conteúdo em transmissões de mídia contínua ao vivo em redes P2P. As soluções propostas são estratégias que não utilizam assinaturas digitais, ou seja, não utilizam criptografia de chave pública, e que também não realizam o envio dos valores *hash* dos *chunks* durante a transmissão.

O modelo de diagnóstico utilizado é o modelo generalizado de diagnóstico baseado em comparações apresentado em [208]. É importante recordar que este modelo permite que a comparação de tarefas executadas por unidades falhas (ou neste contexto, poluídas) pode resultar em igualdade. Além disso, neste modelo, as unidades testadas são classificadas em conjuntos, com base no resultado das comparações. Nas duas soluções propostas, um determinado nodo (ou *peer*) realiza testes através da solicitação de um determinado *chunk* aos seus vizinhos. O resultado da tarefa, que é o conteúdo do próprio *chunk* recebido, é então comparado, em pares. Com base no resultado das comparações os *peers* são agrupados (ou classificados) em conjuntos de acordo com o conteúdo dos *chunks* recebidos.

É importante destacar que os algoritmos empregados pelas duas soluções tanto de detecção como de combate à poluição de conteúdo em transmissões ao vivo – descritas neste capítulo – são diferentes do algoritmo proposto para o diagnóstico de falhas em sistemas de topologia arbitrária – descrito no Capítulo 3. Essa diferença se deve ao fato de que o modelo de diagnóstico do algoritmo proposto no capítulo anterior assume que a comparação de tarefas executadas por unidades falhas resulta em diferença. Por outro lado, em transmissões ao vivo, a comparação de dois *chunks* poluídos, retornados por diferentes *peers*, pode resultar em igualdade.

Ambas as soluções propostas para identificar e combater a poluição foram construídas sobre o protocolo Fireflies [114] – um protocolo escalável que cria uma rede *overlay*. O Fireflies usa a estratégia *pull-based* para a transmissão dos dados e emprega a topologia *mesh* [100]. As implementações foram realizadas usando o mesmo simulador Fireflies descrito em [97].

O restante deste capítulo está dividido em 5 seções. As duas primeiras seções apresentam respectivamente, uma introdução sobre transmissões de mídia contínua ao vivo em redes P2P, e uma apresentação dos trabalhos existentes que investigam o problema da poluição de conteúdo em redes P2P. Na sequência, a terceira seção realiza uma descrição do protocolo Fireflies. Por fim, as duas últimas seções apresentam as duas soluções para identificar e combater a poluição de conteúdo em transmissões ao vivo em redes P2P. A primeira solução – na Seção 4.4 – apresenta uma estratégia baseada em um *tracker* central para a detecção (apenas o diagnóstico) da poluição de conteúdo nas transmissões ao vivo. Já a segunda solução – apresentada na Seção 4.5 – é uma estratégia distribuída e descentralizada com o objetivo de combater a propagação da poluição de conteúdo na rede. Em ambas as soluções propostas, resultados experimentais exaustivos foram realizados e são apresentados nas respectivas seções.

4.1 Transmissões de Mídia Contínua ao Vivo em Redes P2P

Nas transmissões de mídia contínua ao vivo em redes P2P (ou redes *overlay*), um *servidor fonte* é a entidade responsável por gerar e iniciar a disseminação do conteúdo que é transmitido. O conteúdo transmitido é dividido em pedaços chamados *chunks*. O servidor fonte é responsável por enviar estes *chunks* a determinados *peers* da rede. Estes *chunks* são então compartilhados entre os demais *peers* da rede – que são os próprios usuários do serviço – com o objetivo de que todos os *peers* recebam todos os *chunks* transmitidos pelo servidor fonte. Duas topologias são geralmente empregadas para transmissões de conteúdo em redes P2P [137]: a topologia em *árvore* e a topologia *mesh*, descritas a seguir.

Na topologia em *árvore* [51], os *peers* são organizados em forma de árvore, na qual o servidor fonte é o nó raiz. A Figura 4.1 ilustra um exemplo de um servidor fonte e um conjunto de peers dispostos usando a topologia em árvore. As principais vantagens desta topologia é que, após a árvore estar construída, as decisões de transmissão de dados são simples, ou seja, um *peer* recebe dados de seus pais e os repassa aos seus filhos na árvore. Em uma rede que não apresenta falhas esta topologia possui baixo atraso (ou *delay*) na transmissão de dados entre o servidor fonte e os nós folhas da árvore. Por outro lado, esta topologia apresenta algumas desvantagens. Se uma falha ocorrer em um *peer* localizado perto da raiz da árvore, todos os *peers* daquela subárvore – ou seja, os *peers* da árvore que estão abaixo daquele *peer* que falhou – serão afetados. Além disso esta topologia possui baixa resiliência ao *churn*, pois se um *peer* intermediário sair do sistema, toda a sua subárvore precisará ser reconstruída [137]. Por fim, a taxa média de *upload* é menor do que em outras topologias, pois os nós folhas apenas recebem dados e não os retransmitem durante as transmissões.

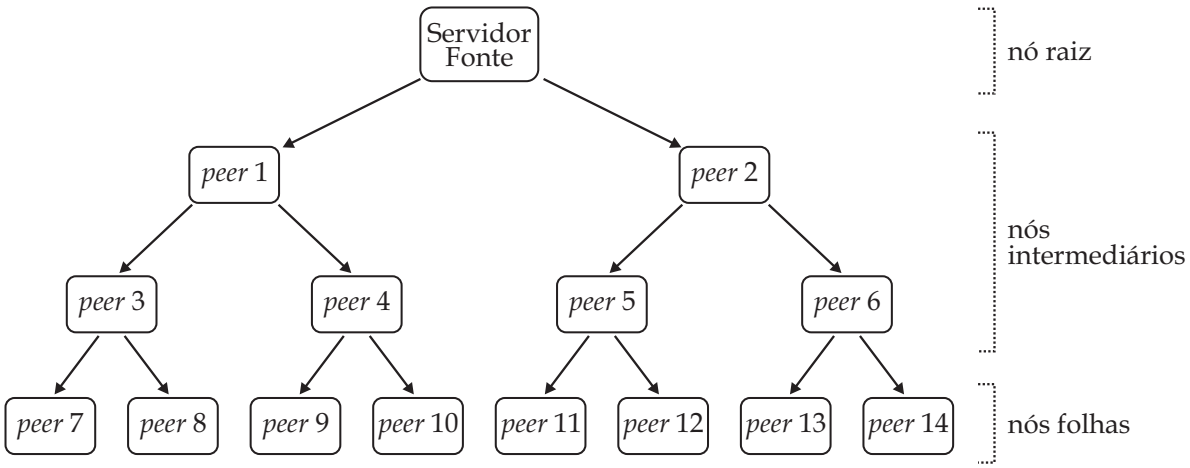


Figura 4.1: Exemplo de uma rede P2P utilizando a topologia em árvore.

Por sua vez, a topologia *mesh* (ou malha) não é estruturada [155], isto é, ela não é baseada em uma estrutura rígida de rede. A Figura 4.2 ilustra um exemplo de uma rede usando a topologia em *mesh*. Nesta figura nota-se que não existe um padrão para a topologia da rede. Nesta topologia, quando um *peer* se junta à transmissão ele simplesmente se conecta a um conjunto de outros *peers* e inicia a troca de informações. O principal

problema das redes em *mesh* está relacionado à forma com a qual os *peers* trocam dados: para receber um determinado *chunk*, um *peer* precisa primeiramente requisitá-lo a outro *peer* que já possui aquela informação; para que isso seja possível, os vizinhos de cada *peer* precisam notificar a disponibilidade daquele *chunk*. Portanto nesta topologia existe um consumo adicional de banda de rede decorrente destas mensagens adicionais.

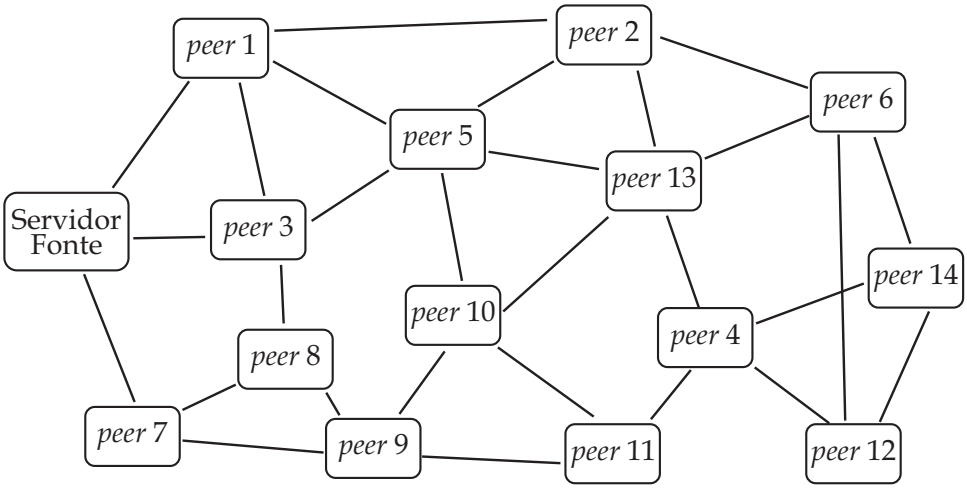


Figura 4.2: Exemplo de uma rede P2P utilizando a topologia em *mesh*.

Em relação à forma específica com que os dados são transmitidos na rede, existem três estratégias comumente utilizadas [137]: a *push-based*, a *pull-based* e a *push-pull-based*. A estratégia *push-based* é usada principalmente pelas topologias em árvore: os dados são transmitidos de um *peer* para outro sem que ele seja solicitado. Apesar dos sistemas que utilizam apenas esta estratégia não possuírem a desvantagem de ter que requisitar os dados, não é possível solicitar novamente um *chunk*, caso ocorra qualquer falha na transmissão. Outra desvantagem desta estratégia é que, se existirem múltiplos transmissores na rede, um determinado *peer* pode receber dados duplicados, o que representa um desperdício no uso de banda de rede.

Na estratégia *pull-based*, um dado específico é enviado por um *peer* a outro apenas se ocorrer a requisição para isso. Além disso, se um determinado *chunk* não foi recebido por qualquer problema, ele pode ser solicitado novamente. Por outro lado, os *peers* trocam mensagens para manter uma série de informações sobre a disponibilidade de *chunks* na

rede e portanto esta estratégia acarreta em um acréscimo de tráfego de rede. Já a estratégia *push-pull-based* [137, 83] combina ambas as estratégias anteriores: os dados são transmitidos sem uma requisição específica, mas um *chunk* específico também pode ser solicitado novamente em caso de perda ou falha na transmissão.

Os sistemas mais populares para a transmissão de mídia contínua ao vivo em redes P2P utilizam a topologia em *mesh* com pedidos explícitos por *chunks*, ou seja, implementam a estratégia *pull-based* para a transmissão de dados [100].

4.2 Poluição de Conteúdo em Redes P2P: Trabalhos Relacionados

Como nas redes P2P os próprios *peers* são responsáveis pela disseminação dos dados que são transmitidos, a poluição de conteúdo – ou *poisoning* [40] – em transmissões ao vivo é um desafio relevante. Além disso soma-se o fato de que atualmente a maioria dos sistemas populares para transmissões de mídia contínua ao vivo em redes P2P não implementam nenhum mecanismo de proteção ou criptografia no envio de suas mensagens [99]. Caso o sistema P2P não adote medidas preventivas para combater ataques de poluição, os participantes da rede podem ser fortemente afetados, mesmo na presença de poucos *peers* maliciosos ou de ataques considerados simples [26, 52, 94].

Em transmissões ao vivo, os ataques de poluição de conteúdo são aqueles em que os participantes maliciosos da rede modificam de forma não autorizada o conteúdo (ou *chunks*) transmitido. A modificação dos *chunks* pode ser de diferentes tipos [91, 53, 135], que incluem: a troca do conteúdo, a criação de novos dados e até a destruição ou omissão dos *chunks* transmitidos. Para combater os ataques de poluição de conteúdo em redes de compartilhamento P2P, diversas técnicas foram criadas – entre elas [134, 191, 185, 202] – e são apresentadas as seguir. No entanto, quando aplicadas para transmissões de mídia contínua ao vivo aumentam a sobrecarga de dados na rede, o que pode ocasionar atrasos nas transmissões e desperdício de banda da rede [184].

As técnicas de listas negras [134] utilizam faixas de IPs para englobar o maior número possível de *peers* que disseminaram conteúdo poluído. Nesta estratégia, os demais *peers* do sistema não enviam nem recebem dados de *peers* que estão na lista negra. O desafio desta técnica é englobar o menor número de *peers* não poluidores nestas faixas de IPs. Por outro lado [91] aponta que esta técnica se mostra custosa quando aplicada para transmissões ao vivo.

Uma estratégia básica para combate à poluição de conteúdo, empregada pelo BitTorrent [52] para compartilhamento de arquivos, é permitir que os *peers* obtenham previamente os valores *hash* (ou *resumos digitais*) de todos os *chunks* [191]. Desta forma quando um *chunk* é recebido, cada *peer* pode verificar a integridade daqueles dados. Esta técnica é eficaz para tratar falhas durante a transmissão dos *chunks* decorrente de falhas físicas nos canais de comunicação. Em transmissões ao vivo um dos problemas dessa técnica está em receber previamente o valor *hash* de conteúdos que são gerados durante a própria transmissão. Mesmo que os valores *hash* sejam gerados pelo servidor fonte e transmitidos juntamente com os *chunks* pela rede, esta técnica ainda permitiria a um *peer* malicioso modificar indevidamente um *chunk* e retransmití-lo juntamente com um novo valor *hash* correspondente.

Outra estratégia consiste na aplicação da criptografia de chave pública, ou seja, disseminar todo *chunk* juntamente com uma assinatura digital correspondente gerada pelo servidor fonte [98, 97]. Uma vantagem desta estratégia é que a assinatura digital pode ser transmitida com o conteúdo do próprio *chunk*, e a assinatura é gerada com a chave privada do servidor fonte. Por outro lado, uma desvantagem desta estratégia é que a verificação das assinaturas digitais em todos os *peers*, para cada um dos *chunks*, pode ser um processo considerado custoso em transmissões ao vivo dependendo dos dispositivos dos usuários envolvidos nas transmissões – e até eventualmente um impeditivo, por exemplo, em casos de dispositivos móveis com recursos limitados. Uma variante desta estratégia, chamada de *Linear Digests* também é apresentada em [98, 97] e agrupa os valores *hash* de um conjunto de *chunks* em uma mesma assinatura digital, que também é gerada pelo

servidor fonte.

Algumas outras ferramentas, como por exemplo em [133, 132], utilizam ainda a criptografia simétrica de todos os *chunks* transmitidos na rede através do estabelecimento de uma chave secreta compartilhada pré-definida. Em [132] também é proposto um mecanismo seguro de gerenciamento de chaves secretas no qual o servidor fonte periodicamente recria e retransmite a nova chave secreta compartilhada para um número limitado de *peers* da rede. Já em [32] é apresentada uma solução que utiliza um grupo responsável por manter a integridade do conteúdo transmitido pelo servidor fonte. Nesta solução o servidor fonte publica o conteúdo a este grupo de *peers*. Cada *peer* que requisita e recebe um dado pela rede, pode verificar a integridade do dado através deste grupo.

Em [185, 202] os autores apresentam soluções baseadas em reputação e *ranking* para sistema P2P de compartilhamento de arquivos. Neste sistema os próprios *peers* da rede classificam outros *peers* como honestos, que por sua vez adquirem acesso ao conteúdo compartilhado. Em [26, 168] os autores apresentam outras soluções baseadas em reputação, mas diferentemente das anteriores, estas soluções são aplicadas para a transmissão de mídia contínua ao vivo. O mecanismo de reputação destas soluções se baseia na própria experiência de cada *peer* e também o consenso da rede. Os autores de [183, 172] enfatizam que tais sistemas de reputação podem sofrer com o conluio de *peers* maliciosos e de falsos positivos, além da demora na convergência do consenso sobre os *peers* da rede P2P. Para desencorajar a troca de identidade e incentivar bons comportamentos, algumas soluções baseadas em reputação ainda consideram *peers* recém chegados ao sistema como suspeitos e diminuem os recursos disponíveis nestes *peers*. Com o mesmo propósito, em [154] se propõe o uso de uma entidade central responsável por identificar os participantes recém chegados ao sistema. Por outro lado, os autores de [81] mostram que esta abordagem central diminui a escalabilidade do sistema.

Outras estratégias alternativas trabalham a redução do custo de autenticação dos *chunks* em transmissões ao vivo. Uma delas são as árvores de Merkle (*Merkle-trees*) [191] onde o servidor fonte calcula o valor *hash* de n *chunks* consecutivos. Estes *hashes* são

então usados como os nós folhas de uma árvore de Merkle e os nós intermediários são identificados pelos valores *hash* de seus filhos na árvore. O valor *hash* de todos os nós nesta estrutura em árvore são combinados para que a autenticação de cada *chunk* seja realizada.

Em [53] os autores realizam a avaliação de quatro técnicas já mencionadas acima: criptografia simétrica, verificação de *hashes*, assinaturas digitais e lista negra. Os autores concluem que o uso de árvores de Merkle é um dos mecanismos mais eficientes em termos da sobrecarga computacional adicionada. Mais recentemente em [135] uma avaliação do impacto de ataques de poluição é apresentada. O trabalho mostra que o impacto de um ataque de poluição não está diretamente relacionado ao tamanho da rede em si, mas depende fortemente dos níveis de *churn* e da banda de rede disponível nos *peers* maliciosos e no servidor fonte.

Uma estratégia baseada em *network coding* [82, 189] chamada MIS (*Malicious node Identification Scheme*) é apresentada em [190] para identificar e limitar a poluição de conteúdo em transmissões ao vivo em redes P2P. Cada *chunk* transmitido pelo servidor fonte é dividido em blocos. Cada bloco é subdividido em palavras (ou *codewords*) – que por sua vez convertem cada um dos *chunks* em uma matriz de elementos de um campo de Galois (*Galois Field*, ou GF). Por fim, blocos codificados (*coded blocks*) – que são criados baseados nas matrizes GF e combinam um vetor de coeficientes aos blocos originais – são as informações transmitidas pelo servidor fonte para os peers. Cada *peer* que recebe os blocos codificados, os decodificam para reconstruir os *chunks* originais.

Em [203] e [150] os autores avaliam a poluição de conteúdo em redes de compartilhamento de arquivos. O primeiro trabalho analisa a poluição de índices e poluição de conteúdo. A análise mostra que três fatores possuem forte impacto na distribuição do conteúdo: a persistência dos arquivos originais, a taxa de falsos positivos, e a situação inicial da rede P2P. Já no segundo trabalho [150] um método é apresentado para quantificar a poluição de conteúdo na rede KAD através da análise do nome dos arquivos e do conteúdo correspondente. Um grande número de arquivos foi avaliado e os resultados

mostram que na amostra avaliada 2/3 dos arquivos estavam poluídos. Em [27, 153] uma caracterização do tráfego gerado no sistema SopCast é realizada. O trabalho observou que um *peer* malicioso foi capaz de comprometer 50% dos *peers* da rede e 30% da largura de banda de *download*.

Em [138] é apresentada uma estratégia para esconder a identidade dos servidores fontes em redes P2P de transmissões de vídeo sob demanda (*P2P Video-on-Demand networks*). Os autores enfatizam que este ponto é relevante pois com a identidade do servidor fonte, um *peer* mal intencionado pode dirigir ataques como o de DDoS a estes servidores com o objetivo de prejudicar a transmissão. Em [110] os autores propõem um esquema de detecção de conteúdo poluído e implementam pedidos de retransmissão dos dados poluídos, para sistemas P2P de transmissões ao vivo. Para identificar a poluição, o trabalho propõe um gerenciamento de confiança, permitindo assim isolar o poluidor através do consenso da rede P2P. Uma desvantagem dessa solução é que o número de retransmissões pode ser elevado, e a convergência do consenso da rede sobre um determinado *peer* é demorada.

Os autores de [91] realizam um *survey* sobre aspectos de segurança e privacidade em redes P2P para transmissões ao vivo. Eles avaliam aspectos como controle de acesso, gerenciamento de identidades, mecanismos de incentivos e punições. Os autores ainda enfatizam que redes que implementam a topologia em árvore são vulneráveis a ataques de poluição. Por fim, recentemente em [43] os autores apresentam uma avaliação dos mecanismos de autenticação de conteúdo em redes P2P para transmissão ao vivo. Os autores comparam a sobrecarga gerada e a segurança adicionada por diversas técnicas e mostram que, para transmissões ao vivo de alta resolução, os mecanismos com sobrecarga aceitáveis avaliados não foram fortemente resilientes aos ataques de poluição.

4.3 O Protocolo Fireflies

O protocolo Fireflies é um protocolo escalável que cria uma rede *overlay* tolerante a intrusões [114]. Todos os *peers* da rede executam o protocolo Fireflies usando a estratégia

pull-based para a transmissão dos dados [137], e os *peers* são organizados em uma topologia *mesh* [155]. Além dos *peers*, também existe um *servidor fonte* que gera os *chunks* que são transmitidos na rede. O servidor fonte é considerado uma unidade confiável e que nunca falha.

No Fireflies os *chunks* são enviados inicialmente pelo servidor fonte para um número limitado de *peers* da rede. Os *peers* então compartilham os *chunks* entre si, com o objetivo de que todos os *peers* da rede recebam todos os *chunks* transmitidos pelo servidor fonte. No protocolo Fireflies todos os *peers* possuem um identificador sequencial e são organizados em múltiplos anéis [114]. O número de anéis é um parâmetro configurável do protocolo e cada anel contem todos os *peers* do sistema. Estes anéis possuem o simples propósito de determinar o conjunto de vizinhos de cada *peer*. É importante ressaltar que é possível que um determinado *peer* possua vizinhos em comum em mais de um anel. Portanto cada *peer* da rede sempre possui pelo menos 2 vizinhos e no máximo $(2 * \lambda)$, onde λ representa o número de anéis configurados.

Como exemplo, a Figura 4.3 ilustra um sistema com 12 *peers* organizados em 4 anéis. Note que os vizinhos do *peer* 1 são os *peers* 3, 4, 5, 6, 7 e 9. A figura também ilustra a situação onde um *peer* possui vizinhos em comum em mais de uma anel: o *peer* 1 possui os *peers* 3 e 9 como vizinhos em dois diferentes anéis. No Fireflies o servidor fonte recebe o identificador 0 e não participa da configuração dos anéis. Os *peers* vizinhos do servidor fonte são determinados de forma aleatória e a quantidade de vizinhos também é definida por um parâmetro configurável do protocolo.

O protocolo Fireflies ainda configura em cada *peer* uma *janela de disponibilidade* e uma *janela de interesse*. A janela de disponibilidade é uma lista que indica quais *chunks* cada *peer* possui disponíveis para envio a seus vizinhos. Já a janela de interesse é uma lista que indica quais *chunks* cada *peer* ainda precisa receber. Quando um *peer* recebe um *chunk*, ele notifica a todos os seus vizinhos sobre a disponibilidade daquele chunk. Com base nestas notificações cada *peer* é capaz de manter uma lista de quais *chunks* estão disponíveis em cada um dos seus vizinhos. Desta forma, se um *peer* i for notificado por

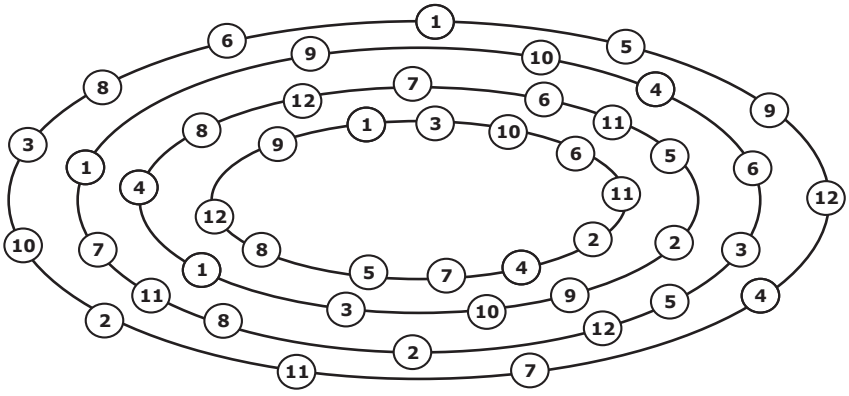


Figura 4.3: Um exemplo de rede Fireflies com 12 *peers* organizados em 4 anéis.

um de seus vizinhos v sobre a disponibilidade de *chunk* c, e se este *chunk* c estiver na janela de interesse do *peer* i, este *peer* requisita o *chunk* c ao vizinho v. Quando o *peer* v receber a requisição, se o *chunk* c ainda estiver na sua janela de disponibilidade, o *peer* v envia o *chunk* c ao *peer* i; caso contrário o *peer* v simplesmente ignora aquela requisição. Este é exatamente o procedimento que também ocorre com todos os *chunks* gerados pelo servidor fonte: quando o fonte gera um novo *chunk* ele notifica seus vizinhos sobre a disponibilidade daquele *chunk*, e então a sua difusão se inicia pelos *peers* da rede.

4.4 Uma Nova Estratégia para o Diagnóstico de Poluição de Conteúdo para Transmissões ao Vivo em Redes P2P

Esta seção apresenta uma nova solução [209, 167] para a detecção de poluição de conteúdo em redes P2P para transmissões de mídia contínua ao vivo. A solução proposta utiliza o diagnóstico baseado em comparações para detectar a poluição de conteúdo e foi construída sobre o Fireflies, que, como descrito na seção anterior, cria uma rede *overlay*. Além do servidor fonte e dos *peers* – que já são componentes da arquitetura do protocolo Fireflies – a estratégia implementa dois novos componentes: o *módulo comparador* e o *tracker*, cujos papéis são descritos a seguir.

O *módulo comparador* é um componente que executa integrado aos próprios *peers* do sistema Fireflies, e tem acesso aos *chunks* recebidos e também às janelas de disponibilidade

e de interesse. Este módulo é responsável por executar a comparação do conteúdo de determinados *chunks*. Os *chunks* que são comparados são obtidos através de requisições a todos os vizinhos de cada *peer*. Os *peers* vizinhos são então classificados em conjuntos de acordo com os resultados das comparações, e esta classificação é enviada ao *tracker*. O *tracker* por sua vez, é uma entidade central confiável, que nunca falha, e é acessível por todos os *peers* da rede. O *tracker* é responsável por receber as classificações enviadas pelos *peers* (através do módulo comparador), consolidá-las em uma única e nova classificação, e, mais importante, realizar o diagnóstico do sistema, ou seja, determinar se há ou não poluição de dados, e quais são as unidades que estão com dados poluídos.

O módulo comparador é executado em cada *peer i* e faz periodicamente a requisição de determinados *chunks* com identificador *cid* (*chunk identifier*) a todos os vizinhos do *peer i*. É importante destacar que toda requisição realizada pelo módulo comparador que executa no *peer i* é direcionada ao próprio sistema Fireflies dos *peers* vizinhos, e o formato destas requisições é idêntico ao de qualquer requisição do sistema Fireflies. Em outras palavras, um *peer* que recebe uma requisição de um módulo comparador – mesmo que seja um *peer* malicioso – não consegue distinguí-las a ponto de tratá-la de forma diferenciada.

Os identificadores dos *chunks* (*cid*) que serão comparados são aleatoriamente determinados pelo *tracker* e repassados ao módulo comparador. Assim que o *peer i* concluir a requisição e receber as respostas com o *chunk* requisitado de identificador *cid* de cada um de seus vizinhos, este *peer i* compara os *chunks* recebidos em pares, e de acordo com o resultado das comparações classifica cada um dos *peers* em um conjunto $U_{i,cid}$. Cada conjunto $U_{i,cid}$ contém o conteúdo de cada um dos diferentes *chunks* recebidos e o identificador dos *peers* que retornaram o *chunk* com aquele exato conteúdo, ou seja, este conjunto possui o seguinte formato:

$$U_{i,cid} = \{(chunk_a, \{peer_i, peer_j, ...\}), (chunk_b, \{peer_k, peer_l, ...\}), ...\}.$$

Um subconjunto específico é criado para relacionar os *peers* vizinhos do *peer i* que não enviaram nenhuma informação sobre o *chunk cid*. Logo que o conjunto $U_{i,cid}$ estiver

completo – ou seja, com informações de todos os *peers* vizinhos do i – este conjunto é enviado ao *tracker*.

Uma otimização foi realizada com o propósito de reduzir o tamanho da mensagem enviada para o *tracker*: o conjunto $U_{i,cid}$ contém o valor *hash* ao invés do próprio conteúdo de cada um dos diferentes *chunks* recebidos, ou seja:

$$U_{i,cid} = \{(hash\_chunk_a, \{peer_i, peer_j, ...\}), (hash\_chunk_b, \{peer_k, peer_l, ...\}), ...\}.$$

Vale lembrar que mesmo sem esta otimização, ou seja, mesmo sem o uso da função *hash*, o funcionamento do algoritmo continua idêntico. Uma asserção é feita sobre o módulo comparador, na qual ele sempre classifica e troca mensagens de forma correta com o *tracker*. Para implementar esta asserção, pode-se utilizar uma abordagem como TSL/SSL [50, 92], na qual criptografia assimétrica é usada no início da sessão e em seguida uma chave secreta é estabelecida para a comunicação entre o *tracker* e o módulo comparador. Outra opção que também pode ser considerada é distribuir o módulo comparador em formato binário [52, 133] com uma chave secreta codificada, que por sua vez será usada como chave inicial de um algoritmo de criptografia simétrica.

A Figura 4.4 mostra um exemplo do funcionamento do sistema. Neste exemplo considera-se que o *chunk* 325 é um dos *chunks* determinados pelo *tracker* para serem comparados. A figura considera que a requisição do *chunk* 325 a partir dos *peers* 19 e 21 para todos os seus vizinhos já foi realizada. A figura mostra o envio do *chunk* 325 por todos os vizinhos destes *peers* 19 e 21. Os rótulos das arestas direcionadas representam o envio de *chunks* que foram requisitados. As demais arestas (não direcionadas) representam os enlaces de comunicação entre os *peers* ou o servidor fonte. Neste exemplo, são mostrados apenas os envios do *chunk* 325 pelos vizinhos dos *peers* 19 e 21, com o propósito de simplificar a figura, mas este mesmo procedimento ocorre com todos os *peers* do sistema.

No exemplo o conteúdo do *chunk* original de identificador 325 é ilustrado por "OrigData", e uma versão poluída do mesmo *chunk* modificada indevidamente pelo *peer* 43 possui conteúdo "PollData". Note que apesar do *peer* 43 ser um *peer* malicioso, neste

exemplo ele enviou uma cópia poluída do *chunk* 325 apenas para o *peer* 21. Considerando este caso, os conjuntos $U_{i,cid}$ após a classificação realizada pelos *peers* 19 e 21 são mostrados na Tabela 4.1. Os próprios *peers* 19 e 21 se incluem nos conjuntos $U_{i,cid}$, sendo inseridos no grupo correspondente ao *chunk* que possuem.

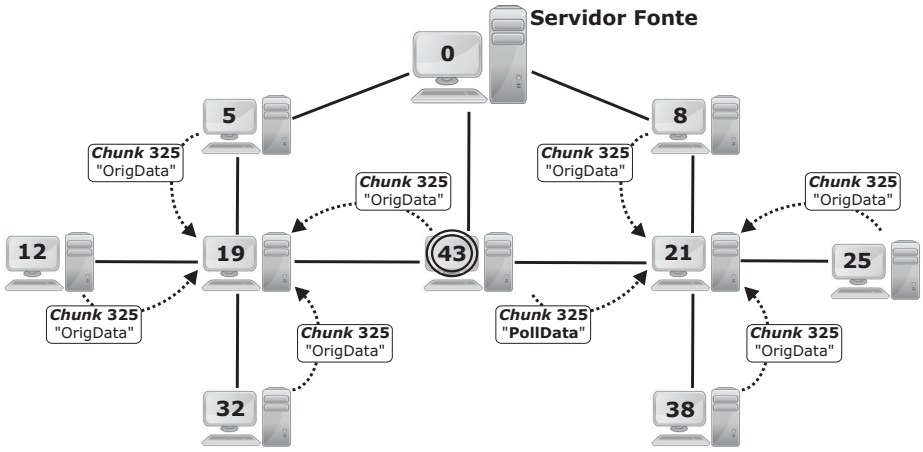


Figura 4.4: Exemplo do envio do *chunk cid* = 325 para os *peers* 19 e 21 por cada um dos seus vizinhos. O *peer* 43 é um *peer* malicioso.

Como o *tracker* recebe de cada *peer i* o conjunto $U_{i,cid}$, o *tracker* terá condições de realizar o diagnóstico completo do sistema e identificar quais unidades estavam com conteúdo poluído. Assim que o *tracker* receber os conjuntos $U_{i,cid}$ de todos os *peers*, o *tracker* realiza uma nova e única classificação de todos os *peers*, agora em um novo conjunto T_{cid}, que por sua vez tem o mesmo formato do conjunto $U_{i,cid}$. Apesar de ter o mesmo formato, neste conjunto T_{cid}, diferentemente do que ocorre nos conjuntos $U_{i,cid}$, um determinado *peer* poderá estar presente em mais de um subconjunto. Um exemplo desta situação também pode ser notada na Figura 4.4, na qual o *peer* 43 enviou *chunks* de conteúdo diferente para os seus *peers* vizinhos 19 e 21. Neste caso o *tracker* irá incluir o *peer* 43 em dois subconjuntos diferentes: no subconjunto indicado por "OrigData" e no conjunto indicado por "PollData".

Como o servidor fonte é confiável e nunca envia diferentes versões de um mesmo *chunk*, no conjunto T_{cid} o fonte estará presente sempre em apenas um único subconjunto. Para realizar o diagnóstico considera-se como falhos, ou seja, com conteúdo diferente do

Peer	*Chunk*	*Conjunto* $U_{i,cid}$
19	325	$U_{19,325} = \{(\text{“}OrigData\text{”}, \{5, 12, 19, 32, 43\})\}$
21	325	$U_{21,325} = \{(\text{“}OrigData\text{”}, \{8, 21, 25, 38\}), (\text{“}PollData\text{”}, \{43\})\}$

Tabela 4.1: Conjuntos $U_{19,325}$ e $U_{21,325}$ gerados respectivamente pelos *peers* 19 e 21.

chunk	*Conjunto* T_{cid}
325	$T_{325} = \{(\text{“}OrigData\text{”}, \{fonte, 5, 8, 12, 19, 21, 25, 32, 38, 43\}), (\text{“}PollData\text{”}, \{43\})\}$

Tabela 4.2: Conjunto T_{325} gerado pelo *tracker* baseado nos conjuntos $U_{19,325}$ e $U_{21,325}$ recebidos.

considerado correto, todos os *peers* que estiverem em mais de um subconjunto e também os *peers* que não estiverem no mesmo subconjunto ao qual o fonte pertence. A Figura 4.5 ilustra o envio dos conjuntos $U_{i,325}$ pelos *peers* 19 e 21 para o *tracker*.

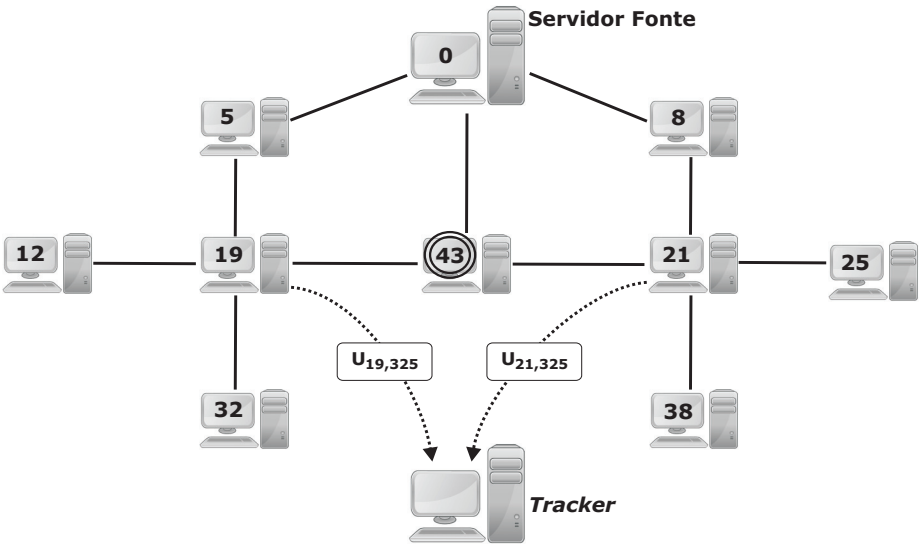


Figura 4.5: Os *peers* 19 e 21 enviam o conjunto $U_{i,325}$ para o *tracker*.

A Tabela 4.2 mostra a classificação final realizada pelo *tracker* para o *chunk* 325. Nesta Tabela um exemplo parcial do conjunto T_{325} é obtido através da junção dos conjuntos $U_{19,325}$ e $U_{21,325}$. Este conjunto T_{325} ainda é parcial, pois o *tracker* continua aguardando os conjuntos $U_{i,325}$ dos demais *peers* do sistema.

Como o módulo comparador executa continuamente, é possível que o *tracker* ainda esteja recebendo por parte de alguns *peers* informações de um determinado *chunk* cid_a enquanto que outros *peers* já estejam enviando informações de outro *chunk* cid_b. Por este motivo, o *tracker* mantém separada e concorrentemente a classificação dos conjuntos T_{cid_a}

```
Algoritmo: ModuloComparador
 1: início
 2:     lista_de_cids ← obter do tracker lista de chunks a serem comparados;
 3:
 4:     sempre que um vizinho v disponibiliza um novo chunk cid faça
 5:       se cid ∈ lista_de_cids então
 6:         se timer_cid não foi inicializado então
 7:           inicializar timer_cid;
 8:         fim se
 9:         obter o chunk cid de v;
10:         atualizar U_{i,cid};
11:       fim se
12:     fim sempre que
13:
14:     sempre que (U_{i,cid} possui dados de todos os vizinhos) ou (timer_cid > limite_resposta) faça
15:       se timer_cid > limite_resposta então
16:         incluir os peers vizinhos que não responderam em conjunto específico de U_{i,cid};
17:       fim se
18:       enviar U_{i,cid} ao tracker;
19:       lista_de_cids ← obter do tracker e atualizar lista de chunks a serem comparados;
20:     fim sempre que
21: fim
```

Figura 4.6: Algoritmo em pseudocódigo do módulo comparador executado em todos os *peers* do sistema.

e T_{cid_b}.

A Figura 4.6 apresenta em pseudocódigo o algoritmo executado pelo módulo comparador. A primeira tarefa executada por este algoritmo, na linha 2, é obter a lista dos *chunks* que serão comparados. Esta informação é obtida através de uma requisição ao *tracker*, que, a cada intervalo de tempo, escolhe aleatoriamente uma lista de *chunks* para ser utilizada como base para as comparações. A partir da lista dos *chunks* que devem ser comparados, o módulo comparador de cada peer permanece, a todo instante, esperando a informação de que algum de seus vizinhos possui um novo *chunk* disponível. O bloco iniciado na linha 4 é executado sempre que algum vizinho v possui um novo *chunk* disponível. Caso o *cid* do novo *chunk* disponível esteja na lista dos *chunks* que devem ser comparados, é verificado se o *timer* daquele *cid* foi inicializado (linha 6). Este *timer* será usado como tempo limite para que o *peer* i realize todas as comparações e a classificação referente ao *chunk cid*. Se este for o primeiro dos vizinhos do *peer* i que esteja disponibilizando um *chunk cid*, o *timer* correspondente ao *chunk cid* é iniciado (linha 7). Na linha 9, ocorre a requisição do *chunk cid* para o *peer* v, e na sequência ocorre a atualização do conjunto $U_{i,cid}$ comparando e classificando o *peer* v de acordo com o conteúdo do *chunk cid* recebido.

O bloco iniciado na linha 14 verifica se o conjunto $U_{i,cid}$ já possui todos os vizinhos

do *peer i* ou se o tempo limite para que os vizinhos enviassem informações sobre aquele *chunk* terminou. Em ambos os casos, o conjunto $U_{i,cid}$ é enviado ao *tracker* (linha 18). No entanto, caso ocorra a existência de vizinhos do *peer i* que ainda não enviaram informações sobre o *chunk cid*, estes vizinhos são classificados em um subconjunto do conjunto $U_{i,cid}$ específico para este propósito (linha 16). O tempo limite de resposta (*limite_resposta*) é um valor relacionado ao tamanho da janela de disponibilidade dos *peers* que também considera a frequência de geração de novos *chunks* do fonte.

Por sua vez o *tracker* executa o algoritmo *Diagnostico* mostrado na Figura 4.7. O *tracker* fica continuamente recebendo os conjuntos $U_{i,cid}$ referentes à classificação realizada pelo *peer i* para o *chunk cid*. Esta ação é mostrada na linha 2 do algoritmo *Diagnostico*. Toda vez que o *tracker* receber um conjunto $U_{i,cid}$, ele classifica os *peers* contidos nos subconjuntos daquele conjunto $U_{i,cid}$ em um novo conjunto T_{cid} (linhas 4–8). Após terminar a classificação para todos os *peers* do sistema ou caso o tempo limite para que os *peers* enviassem os seus respectivos conjuntos $U_{i,cid}$ tenha terminado (linha 12), o *tracker* finaliza e imprime o diagnóstico do sistema (linha 16). Considerando o conjunto T_{cid}, o *tracker* irá considerar como *peers* que possuem conteúdo diferente do considerado correto, todos os *peers* que estiverem em mais de um subconjunto e também os *peers* que não estiverem no mesmo subconjunto ao qual o fonte pertence. Neste algoritmo, se ocorrer o mesmo caso onde um determinado *peer i* não envie o conjunto $U_{i,cid}$ dentro do tempo limite de resposta, este *peer i* é classificado em um subconjunto específico do conjunto T_{cid} (linhas 13 e 14).

4.4.1 Resultados Experimentais: Estratégia de Diagnóstico da Poluição

A estratégia proposta para o diagnóstico de poluição de conteúdo em transmissões ao vivo foi implementada usando o simulador Fireflies descrito em [97]. Experimentos através de simulações foram executados em sistemas com 200, 500 e 1000 *peers*. Cada um dos

```
Algoritmo: Diagnostico
 1: início
 2:     sempre que um conjunto U_{i,cid} for recebido faça
 3:       para todo subconjunto u de U_{i,cid} faça
 4:         se ∃ hash_chunk_u ∈ T_{cid} então
 5:           inserir os peers associados ao hash_chunk_u no grupo correspondente em T_{cid};
 6:         senão
 7:           criar novo subgrupo em T_{cid} com o subconjunto u = (hash_chunk_u, {lista de peers});
 8:         fim se
 9:       fim para
10:     fim sempre que
11:
12:     sempre que (T_{cid} possui dados de todos os peers) ou (timer_cid > limite_resposta) faça
13:       se timer_cid > limite_resposta então
14:         incluir os peers que não responderam em conjunto específico de T_{cid};
15:       fim se
16:       finaliza diagnóstico referente às comparações do chunk cid com base no conjunto T_{cid};
17:     fim sempre que
18: fim
```

Figura 4.7: Algoritmo de diagnóstico executado pelo *tracker*.

experimentos simulou uma transmissão ao vivo por um período de 200 segundos. O servidor fonte gerou 30 *chunks*/segundo e o Fireflies foi configurado para organizar os *peers* em três anéis. O tamanho do *chunk* foi de 10KB. Ambas as janelas de disponibilidade e de interesse de todos os *peers* foram configuradas com 3000 *chunks*. Os experimentos foram executados em um computador com processador AMD Phenom 9500 quad-core x64 e 4GB de memória RAM, executando o sistema operacional Linux 64-*bits*, *kernel* versão 2.6.18-238.el5.

Os principais propósitos dos experimentos foram (a) verificar se os *peers* que receberam conteúdo poluído foram diagnosticados corretamente; (b) calcular a sobrecarga adicionada pela solução proposta em termos da quantidade adicional de *chunks* transmitidos na rede; (c) verificar o comportamento da solução proposta na presença de *churn*; (d) verificar o comportamento da solução para redes de diferentes tamanhos (variando o número de *peers*) e para diferentes intervalos de monitoramento, e (e) avaliar a escalabilidade do *tracker*.

Os principais parâmetros variados nas simulações foram:

(1) O número total de *peers* na rede: variando entre 200, 500 e 1000 *peers*.

(2) A quantidade de *peers* poluidores: foram experimentados 0%, 5%, 10%, 15%, 20% e 25% do número total de *peers*.

(3) A frequência de monitoramento: foram experimentados intervalos de 1

e 15 segundos; este parâmetro influencia a frequência com que o *tracker* escolhe aleatoriamente *chunks* para serem monitorados – que é também a frequência com que o *tracker* produz um novo diagnóstico da poluição no sistema.

(4) O comportamento dos *peers* poluidores, onde dois tipos foram experimentados: (a) modificação do *chunk* com uma probabilidade de 100%, e (b) modificação do *chunk* com uma probabilidade de 50%.

(5) Foram realizados experimentos com e sem *churn*. Para os experimentos com *churn*, duas diferentes configurações foram utilizadas. Nas duas configurações um valor igual a respectivamente 50% e 100% do número inicial de *peers* da rede, foi utilizado para a quantidade de *peers* que entraram e a quantidade de *peers* que saíram do sistema. A entrada dos *peers* na rede seguiu uma distribuição normal com média respectivamente de 50% e 100% do número inicial de *peers* e desvio padrão 20. Já a saída seguiu uma distribuição de *Poisson*, também com média de respectivamente 50% e 100% do número inicial de *peers*.

Foram executados um total de 20.000 experimentos. Os resultados foram sumarizados e são apresentados nos gráficos das Figuras 4.8–4.15. As linhas dos gráficos representam as médias, enquanto que as linhas verticais mostram o intervalo de confiança de 95%.

A Figura 4.8 mostra o número de *chunks* enviados pela rede pelo protocolo Fireflies, sem a solução proposta, para sistemas de 500 peers. Esta figura mostra ambos os resultados de experimentos sem *churn* e também com *churn* (50% de peers). É possível notar que a média do número de *chunks* enviados pelo Fireflies está sempre entre 1,9 e 2,8 milhões de chunks. Em todas as figuras, as linhas dos gráficos identificadas com "sempre" se referem aos experimentos onde os *peers* maliciosos alteram todos os *chunks* transmitidos. Já as linhas identificadas com "aleatório" se referem aos experimentos onde os *peers* maliciosos alteram os *chunks* transmitidos de forma aleatória (ou uniforme), com uma probabilidade

de 50%.

A Figura 4.9 mostra a média do número de *chunks* adicionais requisitados pelo módulo comparador da solução proposta, para sistemas de 500 peers. O módulo comparador foi configurado com intervalo de monitoramento de 15 segundos. É possível notar que a média do número de *chunks* requisitados esteve sempre entre 22.000 e 33.000 chunks. Portanto, em comparação com o número de *chunks* enviados apenas pelo protocolo Fireflies (Figura 4.8), a solução proposta gera uma sobrecarga de cerca de 1,2% de *chunks* adicionais requisitados pelo módulo comparador. Note que esta pequena sobrecarga foi obtida com um intervalo de monitoramento de 15 segundos; dependendo da largura de banda disponível na rede, a frequência com que os *chunks* são monitorados pode ser aumentada. O mesmo vale para o caso contrário, onde esta frequência pode ser diminuída caso exista grande restrição em relação à largura de banda disponível.

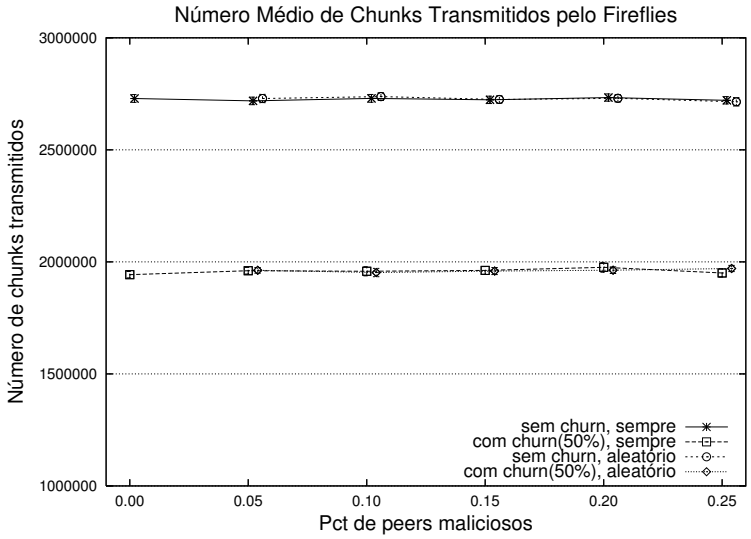


Figura 4.8: Número de *chunks* enviados pelo Fireflies.

A Figura 4.10 mostra a média do número de *peers* que receberam dados poluídos (também para sistemas com 500 peers). Nesta figura nota-se que, mesmo com apenas 5% de *peers* maliciosos, o número médio de *peers* que possuíram *chunks* poluídos em um grupo de experimentos sem *churn* chegou a 86 peers. Já com 25% de *peers* maliciosos, o número médio de *peers* que receberam dados poluídos chegou a 295, o que equivale à 59%

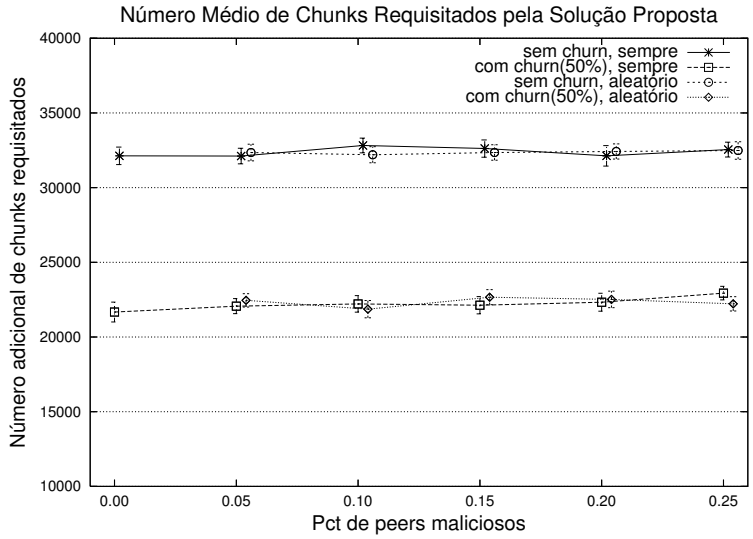


Figura 4.9: Número de *chunks* adicionais requisitados especificamente pelo módulo comparador.

dos *peers* da rede.

A Figura 4.11 mostra a média do número de *peers* que receberam dados poluídos, mas agora em experimentos com diferentes taxas de *churn*. Os experimentos também foram executados em sistemas com 500 peers. Pode-se notar que a média do número de *peers* poluídos com a maior taxa de *churn* de 100% foi na verdade menor do que nos experimentos com taxa de *churn* de 50%, pois nestes experimentos com maior taxa de *churn* um maior número de *peers* maliciosos foram removidos do sistema.

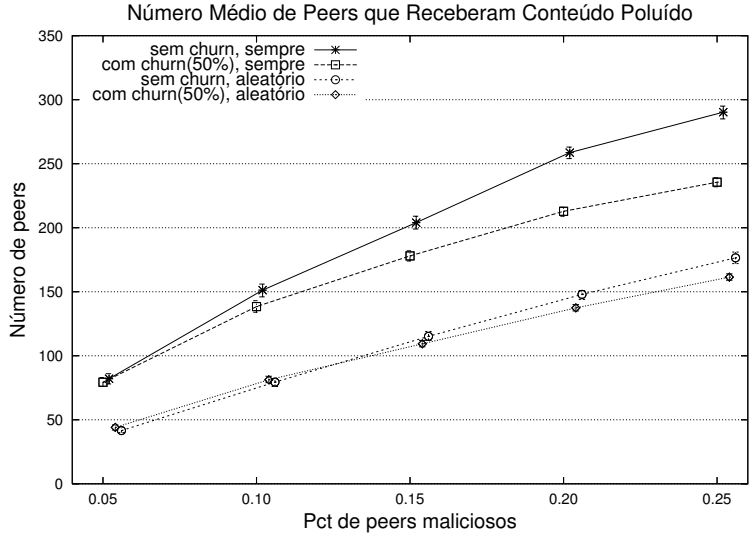


Figura 4.10: Número de *peers* que receberam *chunks* poluídos.

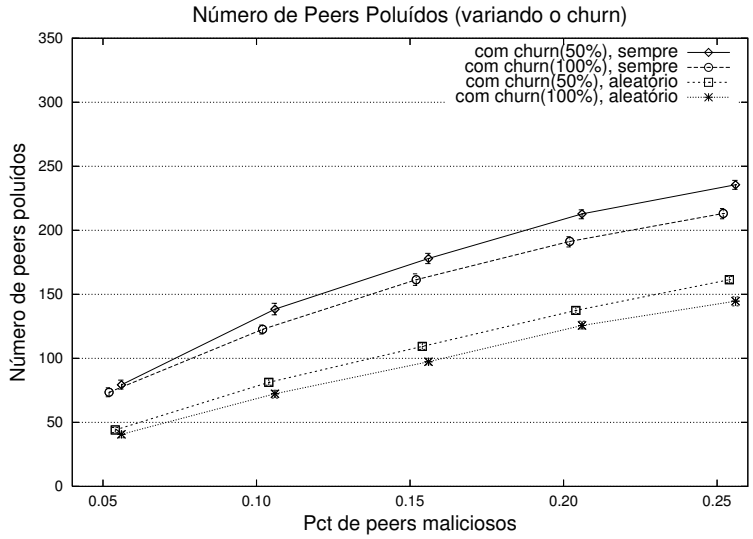


Figura 4.11: Número de *peers* que receberam *chunks* poluídos, variando as taxas de *churn*.

A Figura 4.12 mostra a porcentagem dos *peers* que receberam *chunks* poluídos e que foram diagnosticados corretamente pela solução proposta (também para sistemas de 500 *peers* e intervalo de monitoramento de 15 segundos). A figura mostra que em todos os experimentos a solução proposta identificou corretamente cerca de 95% a 97% dos *peers* que receberam conteúdo poluído. A poluição não foi diagnosticada apenas nos casos onde os *peers* não receberam as respostas dos *chunks* requisitados em tempo – isto é, estes casos são decorrentes da natureza da própria rede P2P, e é uma consequência do tamanho da janela de disponibilidade configurada nos *peers*.

A próxima figura, a Figura 4.13, mostra o número de *chunks* adicionais requisitados pela solução proposta, ou seja, pelo módulo comparador, mas agora variando o número de usuários nos experimentos entre 200, 500 e 1000 peers. A figura sumariza ambos os experimentos executados sem *peers* maliciosos e também os experimentos com 25% de *peers* maliciosos. O intervalo de monitoramento foi de 15 segundos. É possível notar que a quantidade adicional de *chunks* requisitados pela estratégia proposta cresce de forma linear de acordo com o crescimento do número de *peers* na rede.

A Figura 4.14 mostra a média do número de *chunks* requisitados pelo módulo comparador, mas agora variando o intervalo de monitoramento entre 1 e 15 segundos. Em

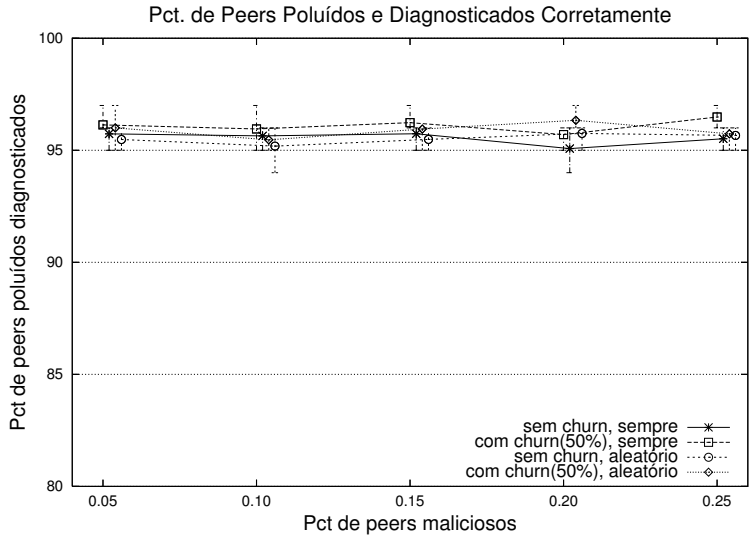


Figura 4.12: Porcentagem dos *peers* poluídos que foram diagnosticados corretamente.

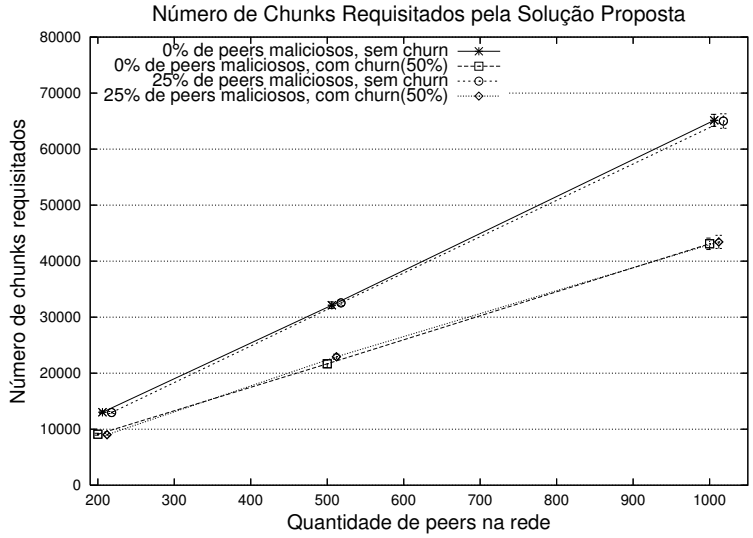


Figura 4.13: Número de *chunks* requisitados pelo módulo comparador, para redes com diferentes quantidades de *peers*.

ambos os casos a rede experimentada foi de 500 peers. Note que o eixo y do gráfico está em escala logarítmica. Os resultados confirmam que o aumento da frequência de monitoramento também aumenta de forma linear a sobrecarga de *chunks* adicionais requisitados.

Finalmente, a Figura 4.15 mostra a média de uso da banda de rede no *tracker* em kbits por segundo, para sistemas de 500 e 1000 *peers* (também para experimentos com intervalo de monitoramento de 15 segundos). O uso da banda de rede esteve abaixo de 1500 kbps a maior parte do tempo, alcançando um pico de 2.8 mbps uma vez após 80 segundos de

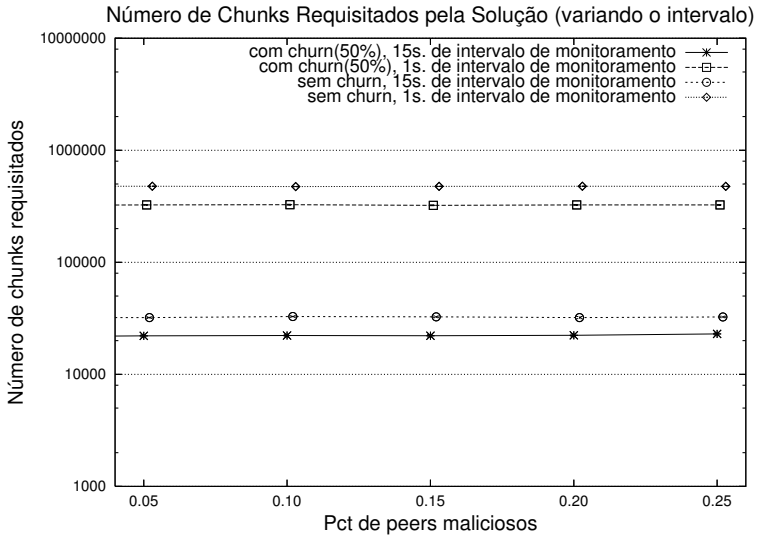


Figura 4.14: Número de *chunks* requisitados pelo módulo comparador, para diferentes taxas de *churn*.

transmissão. Estes resultados podem ser considerados um baixo uso de largura de banda, e mostram que o *tracker* proposto é escalável.

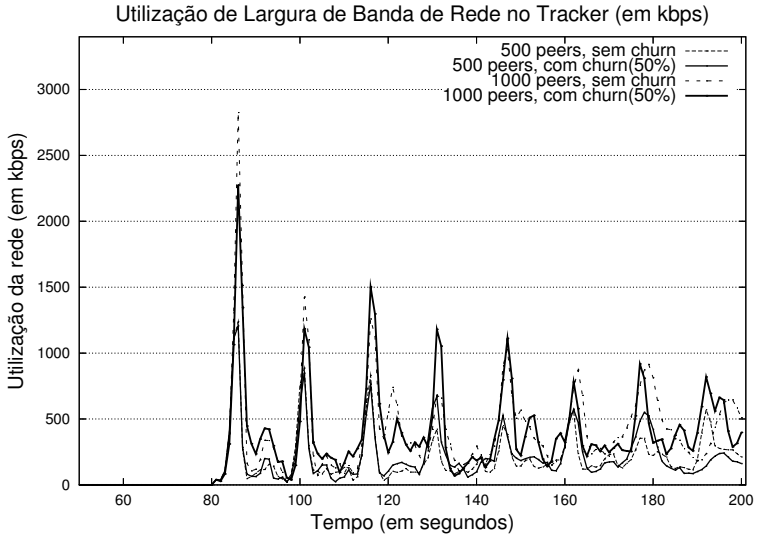


Figura 4.15: Uso da banda de rede do *tracker*.

4.5 Uma Nova Estratégia Completamente Distribuída para Combate à Poluição de Conteúdo em Transmissões ao Vivo em Redes P2P

Esta seção apresenta uma nova estratégia completamente distribuída para o combate à poluição de conteúdo em transmissões ao vivo em redes P2P. Esta solução também utiliza o diagnóstico baseado em comparações para detectar *peers* poluidores e é baseada no protocolo Fireflies. Além disso a estratégia proposta também não utiliza criptografia de chave pública e não utiliza o envio de valores *hash* junto à transmissão.

Diferente da solução apresentada na seção anterior [209, 167] – que realiza apenas o diagnóstico da poluição na rede, sem combatê-la – esta solução tem como principal objetivo combater a propagação da poluição na rede P2P. Além disso, a solução anterior assume a existência de uma unidade central (um *tracker*), enquanto esta é uma estratégia distribuída e descentralizada (ou seja, não utiliza um *tracker*). Nesta solução, cada peer do sistema, de forma independente dos demais, realiza ações com o objetivo de deixar de solicitar *chunks* aos seus vizinhos considerados poluidores. Por outro lado, esta solução não impede a utilização da apresentada na seção anterior, ou seja, mesmo com a utilização da solução proposta nesta seção para combater a propagação da poluição, a solução anterior, que utiliza um *tracker* central, ainda pode ser empregada para realizar o diagnóstico da poluição na rede.

Além do servidor fonte e dos *peers* – que já são componentes da arquitetura do protocolo Fireflies – a solução proposta implementa apenas um novo componente, o *módulo comparador*. O módulo comparador é um componente que é executado por cada *peer*, e é integrado ao próprio código do protocolo Fireflies. Esse módulo possui acesso aos *chunks* recebidos e às janelas de disponibilidade e de interesse do *peer*. Além disso o módulo comparador é o componente responsável por realizar as comparações de determinados *chunks*. Para isso cada *peer*, de forma independente dos demais, escolhe aleatoriamente um *chunk* para ser comparado, dentro dos chamados intervalos de monitoramento. Por sua vez,

o intervalo de monitoramento é uma configuração do módulo comparador que indica o tempo máximo no qual o módulo comparador de cada *peer* deve escolher aleatoriamente um *chunk* para ser comparado.

De forma resumida, o procedimento executado por cada *peer* da rede é o descrito a seguir. Assim que um *peer i* recebe um *chunk* de identificador *cid* de um *peer* vizinho v, este *peer i* verifica se o identificador *cid* é o identificador de um dos *chunks* que foi escolhido aleatoriamente para ser comparado. Caso este seja um dos *chunks* que devem ser comparados, o módulo comparador do *peer i* o requisita a cada um dos vizinhos que informou sua disponibilidade. É importante ressaltar que estas requisições adicionais realizadas pelo modulo comparador, são requisições regulares do protocolo Fireflies, e são realizadas pelo próprio *peer* e através de conexões do próprio sistema. Em outras palavras, qualquer *peer* que receber esta requisição vinda do módulo comparador do *peer i* não consegue diferenciá-la de qualquer outra requisição recebida de qualquer outro *peer*, e portanto não é possível tratá-la de forma diferenciada.

Assim que o *peer i* concluir a requisição e receber as respostas com o *chunk* requisitado com identificador *cid* de cada um de seus vizinhos, este *peer i* compara os *chunks* recebidos em pares, e de acordo com o resultado das comparações classifica cada um dos *peers* em um conjunto $U_{i,cid}$. Cada conjunto $U_{i,cid}$ é um conjunto criado no *peer i* e contém cada um dos *peers* vizinhos do *peer i*, classificados de acordo com o conteúdo do *chunk cid*. Este conjunto possui o seguinte formato:

$$U_{i,cid} = \{(chunk_a, \{peer_j, peer_k, ...\}), (chunk_b, \{peer_m, peer_n, ...\}), ...\}.$$

Como cada *peer* tem um tempo limite para responder a cada requisição de *chunks*, se por qualquer motivo um *peer* não responder a uma requisição sobre um determinado *chunk*, ele é inserido em um conjunto especifico dos *peers* que não responderam às requisições.

Assim que cada *peer i* finalizar um conjunto $U_{i,cid}$, ou seja, o conjunto $U_{i,cid}$ está completo e contém todos os vizinhos do *peer i*, o seguinte procedimento é executado. Se neste conjunto $U_{i,cid}$ existir um subconjunto de *peers* que responderam, e cuja quantidade de *peers* neste subconjunto seja maior que a metade do número de vizinhos do *peer i*, então:

aquele *peer i* (e apenas aquele *peer i*) passa a partir daquele momento a ignorar todos os *chunks* e notificações de disponibilidade de *chunks* de qualquer um dos *peers* que não estiverem neste maior conjunto. Esta lista de *peers* bloqueados é constantemente atualizada por cada um dos *peers* do sistema, sempre que cada módulo comparador finalizar um novo conjunto U. Como esta lista é periodicamente atualizada, a solução proposta permite a reabilitação de *peers*, ou seja, caso um determinado *peer* seja bloqueado por se comportar como um poluidor, se o seu comportamento mudar, ele pode sair da lista de *peers* bloqueados nos próximos períodos de comparações.

Como exemplo, a Figura 4.16 ilustra as requisições realizadas pela estratégia proposta. Este exemplo considera que o *peer* 20 escolheu o *chunk* com identificador 325 como um dos *chunks* para serem comparados. Como os vizinhos do *peer* 20 na rede são os *peers* 5, 12, 32, 43 e 57, assim que estes *peers* vizinhos notificarem ao *peer* 20 que possuem o *chunk* 325 disponível, o *peer* 20 irá solicitar a cada um de seus vizinhos este *chunk* 325. Nesta figura as setas direcionadas representam o envio do *chunk* 325 para o *peer* 20, a partir de cada um dos seus vizinhos; as demais arestas representam *links* de comunicação entre os próprios *peers* ou entre os *peers* e o servidor fonte. Ainda neste exemplo, o conteúdo original do *chunk* 325 é ilustrado por "OrigData" e o conteúdo modificado (poluído) indevidamente deste *chunk* é ilustrador por "PollData". O exemplo considera que apenas o *peer* 43 é um *peer* poluidor e que neste momento os *peers* 5, 12, 32 e 57 possuem uma cópia correta do *chunk* 325.

Ainda com base na Figura 4.16, assim que o *peer* 20 receber o *chunk* 325 de todos os seus vizinhos, o *peer* 20 irá realizar as comparações dos *chunks* recebidos, em pares, e classificará cada um dos seus *peers* vizinhos no conjunto $U_{20,325}$. Especificamente neste exemplo o conjunto gerado, após finalizado, será $U_{20,325} = \{(OrigData, \{5, 12, 32, 57\}), (PollData, \{43\})\}$. Assim que este conjunto $U_{20,325}$ for finalizado pelo *peer* 20, será possível identificar que existe um conjunto com mais da metade do número de vizinhos do *peer* 20. Em outras palavras, o conjunto $(OrigData, \{5, 12, 32, 57\})$ possui 4 *peers*, e $4 \geq (5/2)$, onde $5/2$ é a metade do número de vizinhos do *peer* 20. A

partir deste momento o *peer* 20 irá deixar de solicitar *chunks* ao *peer* 43. Por outro lado, é importante destacar que caso a maior parte dos vizinhos de um determinado *peer i* sejam *peers* maliciosos e atuem em conluio, ou ainda se a maior parte destes vizinhos se comportem como poluidores passivos – ou seja, constantemente repassem conteúdo poluído – a solução proposta não conseguirá impedir que o peer *i* seja afetado pela poluição de conteúdo.

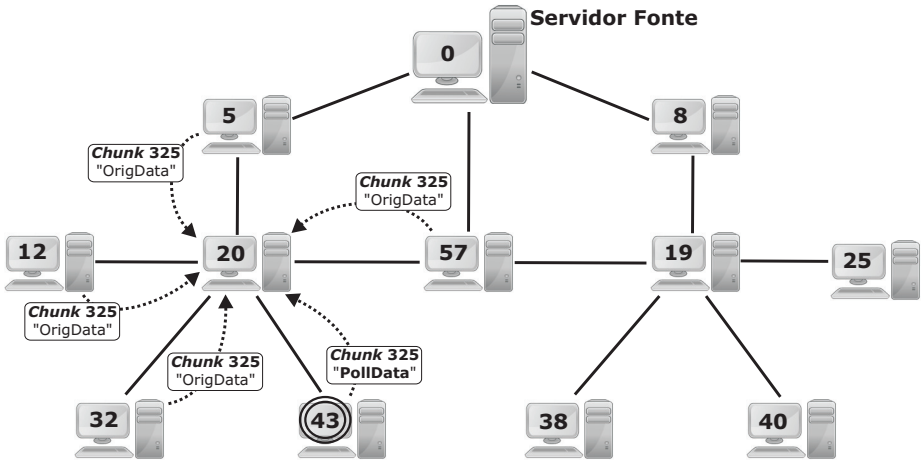


Figura 4.16: Transmissão do *chunk* 325 para o *peer* 20; o *peer* 43 é poluidor.

De forma detalhada e complementando o resumo apresentado acima, as Figuras 4.17 e 4.18 mostram os algoritmos em pseudocódigo que implementam a solução proposta. A Figura 4.17 mostra o algoritmo em pseudocódigo executado pelo módulo comparador, ou seja, é o código responsável pelas solicitações adicionais de *chunks*, pela realização das comparações e pela geração da lista de *peers* vizinhos que terão notificações de *chunks* ignoradas – a lista dos *peers* bloqueados.

Inicialmente (na linha 2) o módulo comparador que executa no *peer i* determina aleatoriamente os identificadores dos primeiros *chunks* para serem comparados por aquele *peer i*. Os identificadores de *chunks* para comparação são escolhidos aleatoriamente, da seguinte forma: $proximo\_cid\_para\_comparar \leftarrow ultimo\_cid\_gerado + mod(random, (monitoring\_interval * mcast\_rate))$, onde $random$ representa um número aleatório, $monitoring\_interval$ é o tempo máximo (em segundos) configurado como o intervalo de monitoramento da solução, e $mcast\_rate$ é o número de *chunks* gerados

por segundo pelo fonte. Em outras palavras, o próximo *chunk* a ser monitorado (ou comparado) será qualquer um dos *chunks* gerados pelo servidor fonte nos próximos *monitoring_interval* segundos após o momento de geração do último *chunk*. Como exemplo, caso $monitoring\_interval = 15$ segundos, $mcast\_rate = 30$ chunks/segundo e $ultimo\_cid\_gerado = 5674$, o valor do *proximo_cid_para_comparar* será um número aleatório entre 5674 e $(5674 + (15 * 30))$. Assim que os próximos identificadores para comparação forem sendo escolhidos eles são inseridos na lista *lista_de_cids*.

```
Algoritmo: ModuloComparador  /* executando no peer i */
 1: inicio
 2:     lista_de_cids ← gera e atualiza a lista de chunks aleatórios para serem comparados
 3:                     (considerando o intervalo de monitoramento configurado);
 4:     sempre que um vizinho v disponibilizar um novo chunk cid faça
 5:       se cid ∈ lista_de_cids então
 6:         requisitar o chunk cid ao peer v;
 7:         atualizar o U_{i,cid} correspondente com a resposta do peer v;
 8:       fim se
 9:     fim sempre que
10:
11:     sempre que ((U_{i,cid} possuir informação sobre todos os peers vizinhos do i) ou
12:                 (acabou o tempo limite para obtenção de informações sobre aquele chunk cid)) faça
13:       se (acabou o tempo limite para obtenção de informações sobre aquele chunk cid) então
14:         incluir vizinhos que não responderam em um subconjunto específico do U_{i,cid};
15:       fim se
16:       se U_{i,cid} possui um subconjunto com mais de N(i)/2 de peers então
17:         lista_de_peers_bloqueados ← ∅; /* limpa a lista de peers bloqueados */
18:         lista_de_peers_bloqueados ← peers que não estão no maior conjunto;
19:       fim se
20:       list_de_cids ← atualiza a lista com novos chunks aleatórios para serem comparados;
21:     fim sempre que
22: fim
```

Figura 4.17: Algoritmo em pseudocódigo implementado pelo módulo comparador.

```
Algoritmo: Peer
 1: inicio
 2:     ...
 3:     sempre que um peer vizinho v notificar a disponibilidade de um novo chunk cid faça
 4:       se cid ∈ lista_de_peers_bloqueados então
 5:         ignorar notificação do peer v;
 6:       senão
 7:         executar o código previsto pelo protocolo Fireflies;
 8:       fim se
 9:     fim sempre que
10:     ...
11: fim
```

Figura 4.18: Parte de código adicionado ao código dos *peers*.

O módulo comparador então espera por notificações dos seus vizinhos sobre a disponibilidade de novos *chunks*. Sempre que um vizinho v notificar a disponibilidade de requisição de um novo *chunk cid* (linha 4 da Figura 4.17), o *peer i* executa o código das linhas 5–8: se o *chunk* de identificador *cid* é um *chunk* para ser monitorado, uma requisição

do *chunk cid* é realizada pelo *peer i* ao *peer v* (linha 6). Assim que receber a resposta da requisição, ou seja, uma cópia do próprio *chunk cid* enviada pelo *peer v*, o conjunto $U_{i,cid}$ é atualizado (linha 7) e o *peer v* é classificado de acordo com o resultado da comparação do *chunk v* recebido.

Já as linhas 11–21 são executadas sempre que o conjunto $U_{i,cid}$ estiver completo, ou seja, já possuir informações sobre todos os vizinhos do *peer i*, ou ainda se o tempo limite para obtenção de informações sobre o *chunk cid* estiver esgotado. Caso tenha ocorrido este último caso (o tempo limite foi esgotado), os *peers* que não enviaram nenhuma informação a respeito do *chunk cid* são classificados em um subconjunto específico (linha 14). Por sua vez, o teste da linha 16 verifica se o conjunto $U_{i,cid}$ em questão possui algum subconjunto com mais de $N(i)/2$ *peers* que responderam, onde $N(i)$ é o número de vizinhos do *peer i*. Caso ocorra esta condição, o módulo comparador atualiza a lista *lista_de_peers_bloqueados* (linhas 17–18) que é usada pelo *peer i* como a lista de *peers* dos quais o *peer i* irá a partir daquele momento ignorar a notificação sobre a disponibilidade de novos *chunks*. A lista *lista_de_peers_bloqueados* é sempre atualizada com todos os *peers* que não estão no maior subconjunto de *peers*.

Finalmente a Figura 4.18 mostra um pequeno trecho do código que foi adicionado ao código de cada *peer* do sistema Fireflies. Cada *peer* do sistema deve realizar um simples teste adicional para verificar se as notificações de disponibilidade de *chunks* devem ou não ser ignoradas. Para isso, sempre que um vizinho *v* notificar a disponibilidade de um novo *chunk*, o *peer i* verifica se este vizinho *v* está na lista *lista_de_peers_bloqueados* (linha 3 da Figura 4.18). Em caso afirmativo, o *peer i* simplesmente descarta aquela notificação. Caso contrário, o *peer i* executa os procedimentos previstos pelo protocolo Fireflies para aquele evento da rede *overlay*.

4.5.1 Resultados Experimentais: Estratégia de Combate à Poluição

A estratégia proposta foi implementada usando o simulador Fireflies descrito em [97]. Um grande número de simulações foi executado para sistemas com 200 *peers*. Cada um dos experimentos simulou uma transmissão ao vivo por um período de 180 segundos. O servidor fonte gerou 30 *chunks*/segundo e o Fireflies foi configurado para organizar os *peers* em 15 anéis. O tamanho do *chunk* foi de 10KB. Ambas as janelas de disponibilidade e de interesse de todos os *peers* foram configuradas com 3000 *chunks*. Além disso, em todos os experimentos o intervalo de monitoramento configurado para o módulo comparador foi de 15 segundos, isto é, no máximo a cada 15 segundos o módulo comparador de cada *peer* escolhia aleatoriamente um *chunk* para ser comparado entre todos os seus vizinhos. Os experimentos foram executados em um computador com processador AMD Phenom 9500 quad-core x64 e 4GB de memória RAM, executando o sistema operacional Linux 64-*bits*, *kernel* versão 2.6.18-238.el5.

Os principais propósitos dos experimentos foram (a) verificar qual foi o impacto da poluição nas transmissões, para diferentes quantidades de *peers* poluidores, e (b) calcular a sobrecarga adicionada pela solução proposta, em termos do número de *chunks* adicionais requisitados pelo módulo comparador. Os principais parâmetros variados nas simulações foram:

(1) a quantidade de *peers* poluidores, variando entre 0%, 5%, 10%, 15%, 20% e 25% do total de *peers* na rede;

(2) foram experimentadas simulações com e sem *churn*: para os experimentos com *churn*, 100 *peers* entraram na rede e outros 100 *peers* saíram da rede. O momento de entrada dos *peers* seguiu uma distribuição normal com média 100 e desvio padrão 20. Para modelar a saída dos *peers* da rede foi utilizada uma distribuição de Poisson com média 100; e

(3) também foram realizadas simulações com o módulo comparador ativo e

inativo, com o objetivo de comparar o efeito da poluição em redes com a solução proposta, e em transmissões sem nenhuma solução para tratar a poluição.

Foram executadas um total de 1.000 simulações. Os resultados foram sumarizados e são apresentados nos gráficos das Figuras 4.19 a 4.27. As linhas dos gráficos representam os valores médios, enquanto que as linhas verticais mostram o intervalo de confiança de 95% para a amostra de dados correspondente.

A Figura 4.19 mostra o número médio de *chunks* enviados normalmente pelo Fireflies durante as transmissões, sem a solução proposta, para as simulações com *churn* e sem *churn*. Pode-se notar que o número médio de *chunks* enviados pelo Fireflies esteve sempre entre 1 e 1,15 milhões de *chunks*. Já a Figura 4.20 mostra o número médio de *chunks* adicionais requisitados pela solução proposta, isto é, por todos os módulos comparadores de todos os *peers*. Pode-se notar que o número de *chunks* adicionais requisitados esteve sempre em torno de 70.000 a 100.000 *chunks*.

A Figura 4.21 mostra a proporção entre as informações das duas figuras anteriores, ou seja, mostra a percentagem do *overhead* gerado pela solução proposta, em termos da quantidade de *chunks* adicionais requisitados pelos módulos comparadores. A figura mostra que o módulo comparador adiciona uma quantidade de 7% a 8% de requisições adicionais de *chunks* em relação ao número de *chunks* já enviados normalmente pelo Fireflies na rede. Em outras palavras, a solução proposta adicionou uma sobrecarga de 7% a 8% ao tráfego de rede. É importante lembrar que o intervalo de monitoramento configurado foi de até 15 segundos, e dependendo da largura de banda de rede disponível esta frequência pode ser aumentada ou diminuída.

As Figuras 4.22 e 4.23 mostram a quantidade média de *chunks* poluídos durante toda a transmissão para redes sem a solução proposta e para transmissões com a solução proposta implementada. Pode-se notar que para experimentos sem *churn* – Figura 4.22 – com a solução proposta implementada, o percentual médio de *chunks* poluídos durante toda a transmissão caiu de 27.1% para 1% em experimentos onde 20% dos *peers* da rede foram

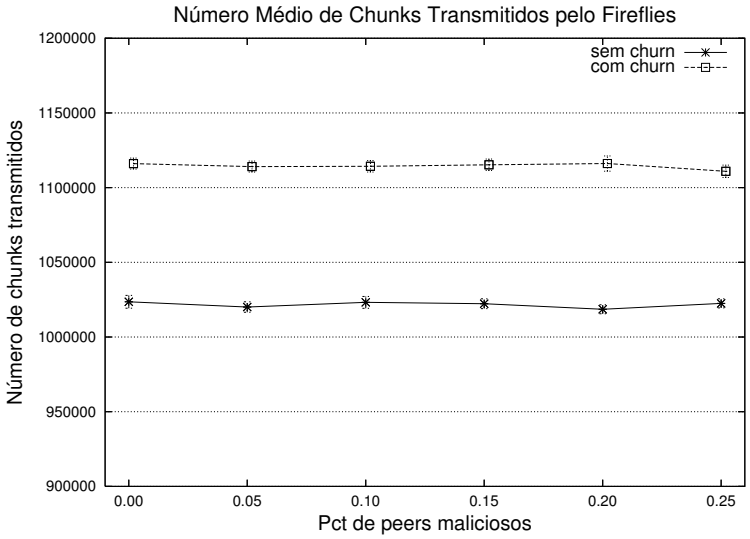


Figura 4.19: Número de *chunks* transmitidos normalmente pelo Fireflies.

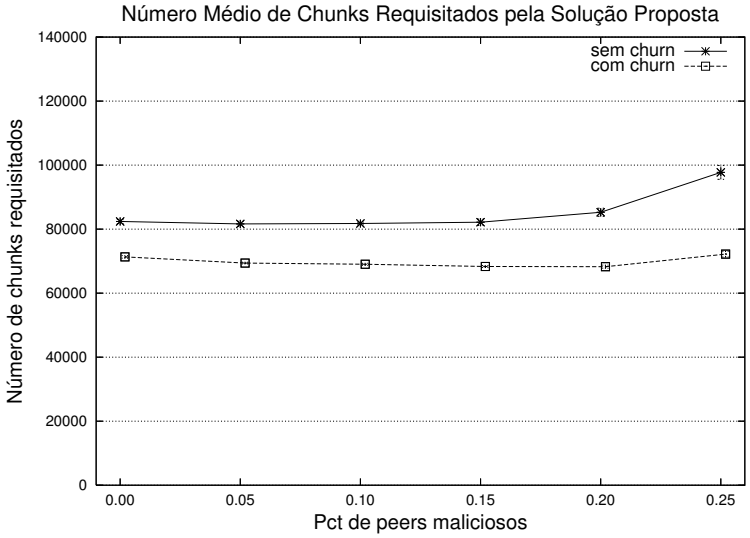


Figura 4.20: Número de *chunks* adicionais requisitados pelo módulo comparador.

configurados como poluidores, e caiu de 33.3% para 5.3% nos experimentos onde 1/4 dos *peers* da rede eram *peers* poluidores. Já para experimentos com *churn* – Figura 4.23 – o percentual de poluição caiu de 45.7% para 7% em experimentos onde 20% dos *peers* eram poluidores, e caiu de 52% para 16% nos experimentos onde 25% dos *peers* eram poluidores.

As próximas quatro figuras mostram o percentual da quantidade de *chunks* poluídos, também para redes com e sem a solução proposta, mas agora durante cada segundo do

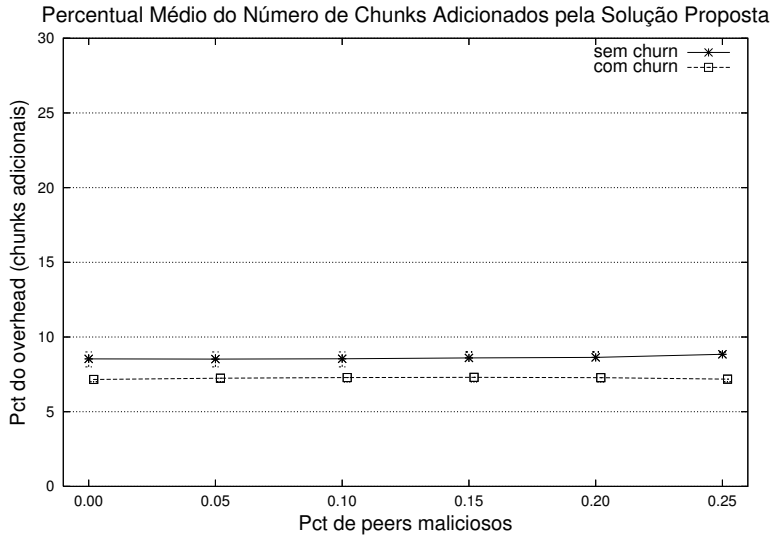


Figura 4.21: Percentual do *overhead* de *chunks* adicionados pela solução proposta.

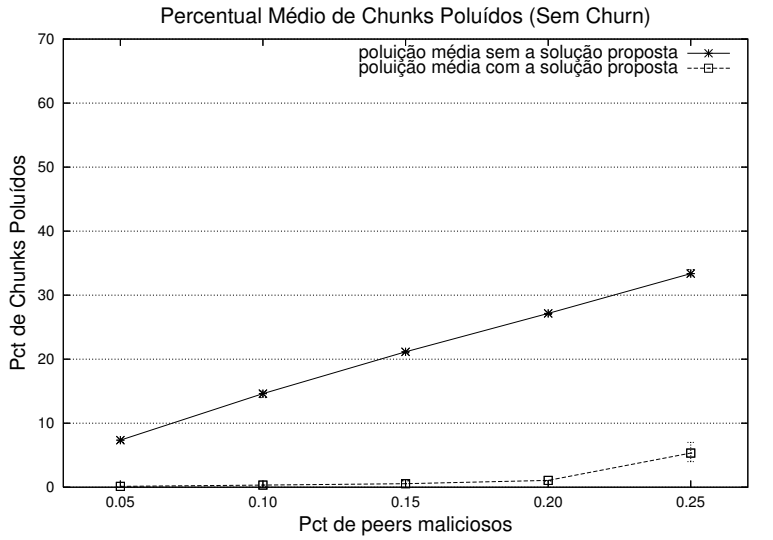


Figura 4.22: Percentual médio de *chunks* poluídos durante as transmissões em sistemas sem *churn*.

tempo das transmissões: as Figuras 4.24 e 4.25 mostram a poluição em redes sem a solução proposta, e as Figuras 4.26 e 4.27 mostram a poluição durante as transmissões com a solução proposta ativa.

Em transmissões sem *churn* e sem a solução proposta – Figura 4.24 – pode-se notar que o percentual de *chunks* poluídos tem valores diferentes para cada quantidade de *peers* poluidores na rede, mas de forma geral possui dispersão pequena durante toda a transmissão. Já para as simulações com *churn* – Figura 4.25 – pode-se notar um aumento

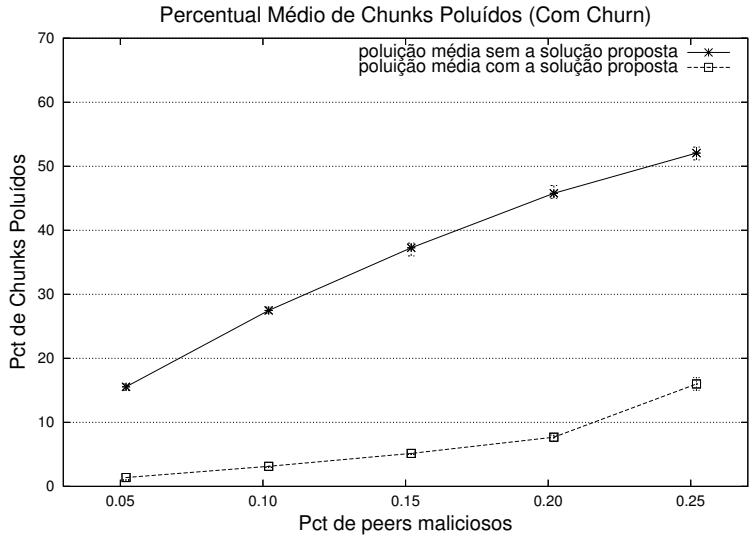


Figura 4.23: Percentual médio de *chunks* poluídos durante as transmissões em sistemas com *churn*.

do percentual de *chunks* poluídos, conforme o tempo de simulação passa da metade do tempo de transmissão e é quando o *churn* aumenta – chegando a percentuais próximos de 70% da quantidade de *chunks* poluídos, em transmissões com 25% de *peers* poluidores.

Com base na Figura 4.26 (experimentos sem *churn*) pode-se notar que, nas transmissões com a solução proposta ativa, a poluição praticamente acabou após cerca de 40 segundos de transmissão para as simulações com até 20% de *peers* configurados como poluidores. Nas simulações com 25% de *peers* poluidores, o percentual de *peers* poluídos no sistema caiu para cerca de 2% após 80 segundos de transmissão. Casos como este das simulações com 25% de *peers* poluidores onde a poluição ainda continuou com valores acima de 0% ocorrem pois o aumento da quantidade de *peers* poluidores na rede aumenta também a probabilidade da maior parte dos vizinhos de um determinado *peer* serem *peers* poluidores.

A Figura 4.26 ainda mostra que a poluição no sistema foi mais alta apenas nos segundos iniciais das transmissões, ou seja, após as primeiras comparações serem realizadas pelos módulos comparadores, cada *peer* foi capaz de identificar e não solicitar mais dados aos *peers* identificados como poluidores. Além disso, também com base nestes dados, se os 30 ou 40 segundos iniciais de cada transmissão fossem retirados do cálculo da média, o

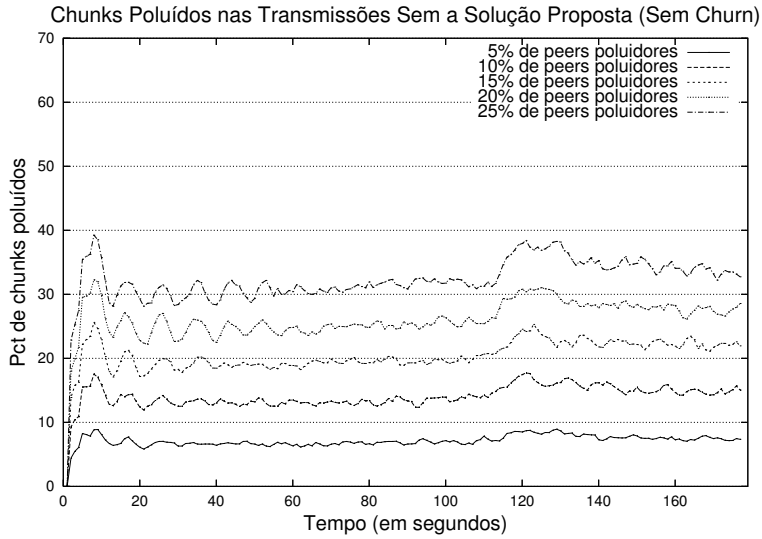


Figura 4.24: *Chunks* poluídos transmitidos em redes sem a solução proposta, em cada segunda da transmissão. Experimentos sem *churn*.

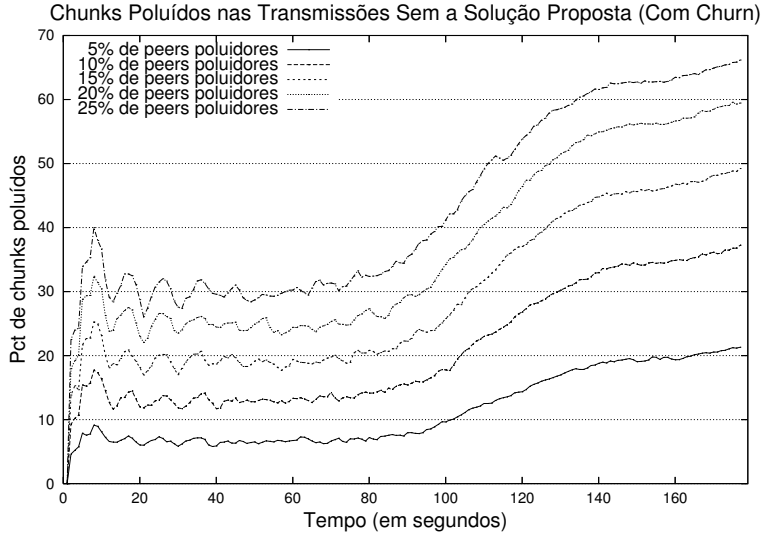


Figura 4.25: *Chunks* poluídos transmitidos em redes sem a solução proposta, em cada segunda da transmissão. Experimentos com *churn*.

percentual de *chunks* poluídos de toda a transmissão – informação que foi mostrada na Figura 4.22 – seria reduzido ainda mais significativamente.

Finalmente com base na Figura 4.27 – que mostra o percentual de poluição nas transmissões com a solução proposta e em experimentos com *churn* – nota-se que nos momentos onde ocorrem as maiores taxas de entrada de novos *peers* na rede (entre os tempos 100 e 120 segundos) ocorre um aumento no percentual de *chunks* poluídos. Este comportamento

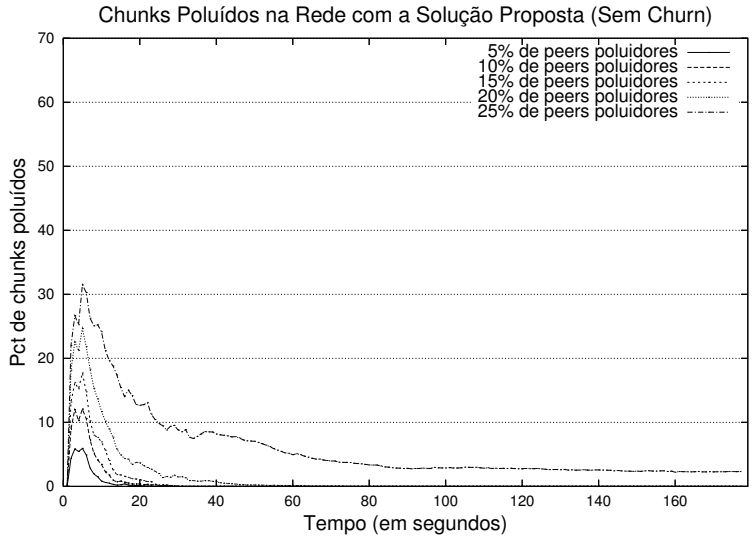


Figura 4.26: *Chunks* poluídos transmitidos em redes com a solução proposta, em cada segunda da transmissão. Experimentos sem *churn*.

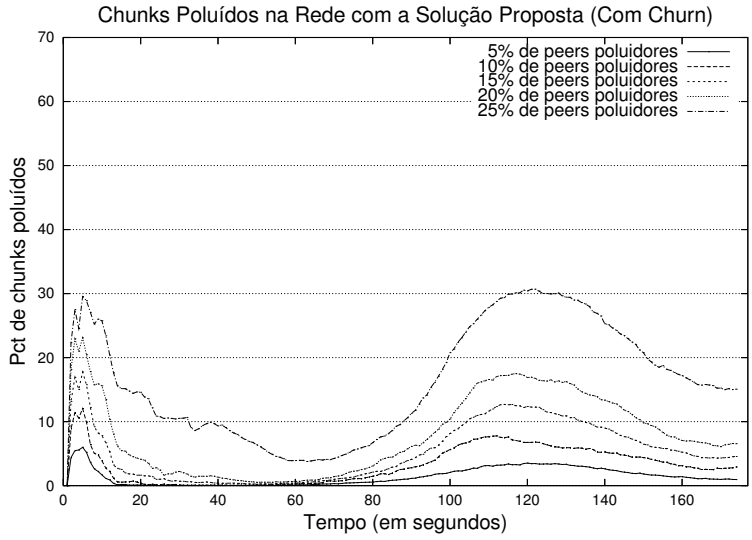


Figura 4.27: *Chunks* poluídos transmitidos em redes com a solução proposta, em cada segunda da transmissão. Experimentos com *churn*.

ocorre pois os novos *peers* que entram na rede não possuem conhecimento inicial sobre quais dos seus vizinhos são *peers* poluidores. Por outro lado, conforme o tempo de simulação avança e estes novos *peers* iniciam as suas comparações, o percentual de poluição volta a cair novamente. Neste sentido, o experimento sem *churn* mostrado na Figura 4.26 reflete melhor o comportamento geral de cada *peer* quando ele inicia sua participação na transmissão, ou seja, nos momentos iniciais, o *peer* acaba recebendo uma maior quan-

tidade de *chunks* poluídos por desconhecer quais dos seus vizinhos são poluidores, mas conforme o tempo avança – e o módulo comparador realiza e finaliza as comparações – este *peer* identifica e para de solicitar *chunks* a vizinhos poluidores.

CAPÍTULO 5

CONCLUSÃO

O diagnóstico baseado em comparações utiliza a comparação do resultado de tarefas produzidos por pares de unidades do sistema. Qualquer diferença na comparação dos resultados das tarefas indica que uma ou ambas as unidades estão falhas. O diagnóstico do sistema é obtido através do conjunto dos resultados de todas as comparações. Este trabalho apresentou um novo algoritmo de diagnóstico baseado em comparações para sistemas de topologia arbitrária, com base no modelo MM*. Foi provado que o novo algoritmo identifica corretamente todas as unidades falhas em sistemas t-diagnosticáveis e que ele possui ordem de complexidade $O(t^2\Delta N)$ no pior caso. Esta complexidade é significativamente menor do que a dos dois outros algoritmos propostos por Sengupta e Dahbura e por Yang e Tang. Como a síndrome de comparações possui tamanho $O(\Delta^2 N)$, a complexidade do algoritmo proposto é muito próxima da complexidade de percorrer os elementos da síndrome uma única vez. Considerando, por exemplo, sistemas completamente conectados, a complexidade do algoritmo proposto é $O(N^3)$ quando $t^2 < N$, e $O(t^2N^2)$ caso contrário – enquanto que nos mesmos casos a complexidade dos dois algoritmos previamente publicados é $O(N^5)$. Este trabalho também apresentou resultados experimentais realizados para sistemas arbitrariamente gerados. Os resultados indicam que na média o número de testes executados pelo algoritmo proposto é cerca de $N^{2.5}$ e ainda que, na maior parte dos grandes sistemas simulados, o algoritmo realiza o diagnóstico executando apenas a parte de seu código que possui complexidade $O(\Delta N^2)$.

Além disso, este trabalho também apresentou duas soluções que utilizam o diagnóstico baseado em comparações para detectar e combater a poluição de conteúdo em transmissões de mídia contínua ao vivo em redes P2P. Ambas as soluções não utilizam criptografia de chave pública e não pressupõe o envio dos valores *hash* dos *chunks* juntamente com a

transmissão. Cada *peer* do sistema executa comparações periódicas sobre determinados *chunks* de seus vizinhos. Com base no resultado das comparações, a primeira solução realiza uma classificação unificada de todos os *peers* poluídos. Já a segunda solução é completamente distribuída e cada *peer* do sistema, de forma independente dos demais, deixa de solicitar *chunks* aos seus vizinhos identificados como poluidores, com o objetivo de combater a propagação da poluição na rede. As soluções foram implementadas no Fireflies, um protocolo escalável para redes *overlay*. Experimentos exaustivos através de simulações foram realizados e mostraram que ambas as estratégias propostas são soluções viáveis para identificar e combater a poluição de conteúdo em transmissões ao vivo e que adicionam baixa sobrecarga ao tráfego da rede. Os resultados ainda mostraram que a solução de combate à propagação da poluição de conteúdo, em diversas configurações foi capaz de reduzir consideravelmente a poluição, em vários casos chegando a eliminá-la no decorrer das transmissões.

5.1 Trabalhos Futuros

Trabalhos futuros incluem uma série de extensões e novos experimentos tanto no algoritmo de diagnóstico baseado em comparações proposto para sistemas de topologia arbitrária como nas soluções de detecção e combate à poluição de conteúdo em transmissões ao vivo. Em relação ao algoritmo de diagnóstico para sistemas de topologia arbitrária com base no modelo MM*, pretende-se avaliar o algoritmo na presença de síndromes parciais e também realizar uma implementação do algoritmo em um sistema distribuído real. O desenvolvimento de um algoritmo de diagnóstico baseado em comparações para sistemas de topologia arbitrária com base no modelo de Chwa e Hakimi [42] também está nos planos de próximos trabalhos.

Em relação às soluções de detecção e combate à poluição de conteúdo em transmissões ao vivo em redes P2P, em um primeiro momento serão realizados novos experimentos com *peers* omissores em ambas as soluções. Também serão realizados esforços para implemen-

tar ambas as estratégias em um serviço real de transmissões ao vivo na Internet. Uma comparação analítica de ambas as soluções com as estratégias que utilizam criptografia de chave pública também está entre os trabalhos futuros. Já sobre a primeira estratégia de diagnóstico de poluição de conteúdo, pretende-se estender a solução para permitir a utilização de múltiplos *trackers* com o objetivo de tolerar falhas sobre este componente.

Mais especificamente em relação à solução de combate à poluição, será realizada uma avaliação da solução utilizando intervalos de monitoramento dinâmicos. Uma avaliação do ponto de convergência entre sobrecarga adicionada, percentual de poluição e frequência de monitoramento também será realizada com a solução de combate à poluição. Uma avaliação de estratégias baseadas em comparações para combate à poluição de conteúdo em outros tipos de redes *overlay* também está prevista.

Além disso, uma investigação de aplicações em engenharia de software [159] também será realizada, como por exemplo em testes de mutação, testes de perturbação e regressão. Por fim, outras possíveis aplicações ainda incluem a criação de sistemas tolerantes a intrusões [131, 161], e o diagnóstico de falhas em sistemas multi-tarefas (*multithreads*) baseados em processadores com múltiplo núcleos [116, 17, 90].

PUBLICAÇÕES REALIZADAS NO DOUTORADO

A lista a seguir registra ambos os artigos publicados e também os artigos submetidos durante o período deste doutorado.

— Elias P. Duarte Jr., Roverli P. Ziwich and Luiz C. P. Albini. A Survey of Comparison-Based System-Level Diagnosis. *ACM Computing Surveys*, Vol. 43, Issue 3, pp. 22:1–22:56, Abr. 2011.

— Roverli P. Ziwich, Emanuel A. Schimidt, Elias P. Duarte Jr. and Ingrid Jansch-Pôrto. Diagnosis of Content Pollution in P2P Live Streaming Networks. *Latin-American Symp. on Dependable Computing (LADC'2013)*, Rio de Janeiro, RJ, Brazil, pp. 48–57, Abr. 2013.

— Emanuel A. Schimidt, Roverli P. Ziwich, Elias P. Duarte Jr. and Ingrid Jansch-Pôrto. Diagnóstico de Poluição de Conteúdo em Redes P2P para Transmissões de Mídia Contínua ao Vivo. *Proc. of the 17th Brazilian Symp. on Multimedia and the Web (WEBMEDIA'2011)*, Florianópolis, SC, Brazil, pp. 221–228, Out. 2011.

— Roverli P. Ziwich and Elias P. Duarte Jr. Uma Nova Estratégia para o Diagnóstico de Falhas Baseado em Comparações. *10° Workshop de Testes e Tolerância a Falhas (WTF'2009), Anais do WTF/LADC'2009*, João Pessoa, PB, Brazil, pp. 76–89, Ago. 2009.

— Roverli P. Ziwich, Glaucio P. Silveira and Elias P. Duarte Jr. Uma Nova Estratégia Completamente Distribuída para Combate à Poluição de Conteúdo em Transmissões ao Vivo. *31° Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2013)*, Brasilia, DF, Brazil, aceito para publicação, Maio 2013.

— Roverli P. Ziwich and Elias P. Duarte Jr. A Nearly Optimal Comparison-Based Diagnosis Algorithm for Systems of Arbitrary Topology. *IEEE Transactions on Computers*, submetido em Fev. 2013.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] K. Abrougui and M. Elhadef. Parallel Self-Diagnosis of Large Multiprocessor Systems Under the Generalized Comparison Model. *Proc. of the 11th Intl. Conf. on Parallel and Distributed Systems*, pages 78–84, July 2005.

[2] S. B. Akers and B. Krishnamurthy. A Group-Theoretic Model for Symmetric Interconnection Networks. *IEEE Transactions on Computers*, 38(4):555–566, Apr. 1989.

[3] L. C. P. Albini, A. Caruso, S. Chessa, and P. Maestrini. Reliable Routing in Wireless Ad Hoc Networks: The Virtual Routing Protocol. *Journal of Network and Systems Management*, 14(3):335–358, Sept. 2006.

[4] L. C. P. Albini, S. Chessa, and P. Maestrini. Diagnosis of Symmetric Graphs Under the BGM Model. *The Computer Journal*, 47(1):85–92, 2004.

[5] L. C. P. Albini and E. P. Duarte Jr. Generalized Distributed Comparison-Based System-Level Diagnosis. *Proc of the 2nd IEEE Latin American Test Workshop*, pages 285–290, Sept. 2001.

[6] L. C. P. Albini, E. P. Duarte Jr., and R. P. Ziwich. A Generalized Model for Distributed Comparison-Based System-Level Diagnosis. *Journal of the Brazilian Computer Society*, 10(3):44–56, Apr. 2005.

[7] J. Amaral, J. Amaral, R. Tanscheit, and M. Pacheco. An Immune Inspired Fault Diagnosis System for Analog Circuits Using Wavelet Signatures. *Proc. of the 2004 NASA/DoD Conf. on Evolvable Hardware*, pages 138–141, June 2004.

[8] E. Ammann and M. Dal Cin. Efficient Algorithms for Comparison-Based Self-Diagnosis. *Self-Diagnosis and Fault Tolerance, Werkhefte der Universitat Ttibingen, 4 Attempto-Verlag, Tubingen*, pages 1–18, 1981.

[9] T. Araki and Y. Shibata. Diagnosability of Networks by the Cartesian Product. *IEICE Transactions on Fundamentals*, E83-A(3):465–470, Mar. 2000.

[10] T. Araki and Y. Shibata. Diagnosability of Butterfly Networks Under the Comparison Approach. *IEICE Transactions on Fundamentals*, E85-A(5):1152–1160, May 2002.

[11] T. Araki and Y. Shibata. Efficient Diagnosis on Butterfly Networks Under the Comparison Approach. *IEICE Transactions on Fundamentals*, E85-A(4), Apr. 2002.

[12] T. Araki and Y. Shibata. (t, k)-Diagnosable System: A Generalization of the PMC Models. *IEEE Transactions on Computers*, 52(7):971–975, July 2003.

[13] A. Bagchi and S. L. Hakimi. An Optimal Algorithm for Distributed System-Level Diagnosis. *Proc. of the 21th IEEE Fault-Tolerant Computing Symp.*, pages 214–221, June 1991.

[14] M. Barborak, A. Dahbura, and M. Malek. The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[15] F. Barsi, F. Grandoni, and P. Maestrini. A Theory of Diagnosability Without Repair. *IEEE Transactions on Computers*, C-25(6):585–593, June 1976.

[16] C. Basile, M. Killijian, and D. Powel. A Survey of Dependability Issues in Mobile Wireless Networks. *Technical Report, Laboratory for Analysis and Architecture of Systems, National Center for Scientific Research, Toulouse, France*, Feb. 2003.

[17] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64-Processor: A 64-Core SoC with Mesh Interconnect. *Proc. of the IEEE Intl. Solid-State Circuits Conf.*, pages 88–598, Feb. 2008.

[18] S. Bettayeb. On the k-Ary n-Cubes. *Theoretical Computer Science*, 140(2):333–339, Apr. 1995.

[19] R. P. Bianchini and R. Buskens. An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation. *Proc. of the 21th IEEE Fault-Tolerance Computing Symp.*, pages 222–229, June 1991.

[20] R. P. Bianchini and R. Buskens. Implementation of On-Line Distributed System-Level Diagnosis Theory. *IEEE Transactions on Computers*, 41(5):616–626, May 1992.

[21] R. P. Bianchini, K. Goodwin, and D. S. Nydick. Practical Application and Implementation of System-Level Diagnosis Theory. *Proc. of the 16th IEEE Fault-Tolerance Computing Symp.*, pages 332–339, June 1990.

[22] D. M. Blough and H. W. Brown. The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation. *IEEE Transactions on Computers*, 48(5):470–493, May 1999.

[23] D. M. Blough and A. Pelc. Complexity of Fault Diagnosis in Comparison Models. *IEEE Transactions on Computers*, 41(3):318–324, Mar. 1992.

[24] D. M. Blough, G. F. Sullivan, and G. M. Masson. Almost Certain Diagnosis for Intermittently Faulty Systems. *Proc. of the 18th IEEE Fault-Tolerant Computing Symp.*, pages 260–271, June 1988.

[25] M. L. Blount. Probabilistc Treatment of Diagnosis in Digital Systems. *Proc. of the 7th IEEE Fault-Tolerance Computing Symp.*, pages 72–77, 1977.

[26] A. Borges, J. Almeida, and S. Campos. Fighting Pollution in P2P Live Streaming Systems. *IEEE Intl. Conf. on Multimedia and Expo*, pages 481–484, Aug. 2008.

[27] A. Borges, P. Gomes, J. Nacif, R. Mantini, J. M. Almeida, and S. Campos. Characterizing SopCast Client Behavior. *Computer Communications*, 35(8):1004–1016, May 2012.

[28] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery Journal*, 2(2):121–167, June 1998.

[29] C.-P. Chang, P.-L. Lai, J. J.-M. Tan, and L.-H. Hsu. Diagnosability of t-Connected Networks and Product Networks Under the Comparison Diagnosis Model. *IEEE Transactions on Computers*, 53(12):1582–1590, Dec. 2004.

[30] C.-P. Chang, T.-Y. Sung, and L.-H. Hsu. Edge Congestion and Topological Properties of Crossed Cubes. *IEEE Transactions on Parallel and Distributed Systems*, 11(1):64–80, Jan. 2000.

[31] G.-Y. Chang, G.-H. Chen, and G. J. Chang. (t, k)-Diagnosis for Matching Composition Networks Under the MM* Model. *IEEE Transactions on Computers*, 56(1):73–79, Jan. 2007.

[32] R. Chen, E. K. Lua, J. Crowcroft, W. Guo, L. Tang, and Z. Chen. Securing Peer-to-Peer Content Sharing Service from Poisoning Attacks. *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing*, pages 22–29, Sept. 2008.

[33] Y. Chen, W. Bucken, and K. Echtle. Efficient Algorithms for System Diagnosis with Both Processor and Comparator Faults. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):371–381, Apr. 1993.

[34] S. Chessa and P. Santi. Comparison-Based System-Level Fault Diagnosis in Ad Hoc Networks. *Proc. of the 20th Symp. on Reliable Distributed Systems*, pages 257–266, Oct. 2001.

[35] C.-F. Chiang and J. J. M. Tan. A Novel Approach to Comparison-Based Diagnosis for Hypercube-Like Multiprocessor Systems. *Intl. Computer Symp.*, pages 166–169, Jan. 2007.

[36] C.-F. Chiang and J. J. M. Tan. A Novel Approach to Comparison-Based Diagnosis for Hypercube-Like Systems. *Journal of Information Science and Engineering*, 24(1):1–9, Jan. 2008.

[37] C.-F. Chiang and J. J. M. Tan. Using Node Diagnosability to Determine t-Diagnosability Under the Comparison Diagnosis Model. *IEEE Transactions on Computers*, 58(1):251–259, Jan. 2009.

[38] Y.-H. Choi and T. Jung. Probabilistic Diagnosis for Sparsely Interconnected Systems. *Proc. of the ACM Annual Conf. on Cooperation*, pages 298–304, Feb. 1990.

[39] S. A. Choudum and V. Sunitha. Augmented Cubes. *Networks Journal*, 40(2):71–84, Sept. 2002.

[40] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks. *Proc. of the 6th ACM Conf. on Electronic Commerce*, pages 68–77, June 2005.

[41] K. Y. Chwa and S. L. Hakimi. On Fault Identification in Diagnosable Systems. *IEEE Transactions on Computers*, C-30(6):414–422, June 1981.

[42] K. Y. Chwa and S. L. Hakimi. Schemes for Fault-Tolerant Computing: A Comparison of Modularly Redundant and t-Diagnosable Systems. *Information and Control*, 49(3):212–238, June 1981.

[43] R. V. Coelho, J. T. Pastro, R. S. Antunes, M. P. Barcellos, I. Jansch-Porto, and L. P. Gaspary. Challenging the Feasibility of Authentication Mechanisms for P2P Live Streaming. *Proc. of the 6th Latin America Networking Conference*, pages 55–63, Oct. 2011.

[44] P. Cull and S. M. Larson. The Möbius Cubes. *IEEE Transactions on Computers*, 44(5):647–659, May 1995.

[45] A. T. Dahbura and G. M. Masson. An $O(n^{2.5})$ Fault Identification Algorithm for Diagnosable Systems. *IEEE Transactions on Computers*, C-33(6):486–492, June 1984.

[46] A. T. Dahbura, K. K. Sabnani, and L. L. King. The Comparison Approach to Multiprocessor Fault Diagnosis. *IEEE Transactions on Computers*, C-36(3):373–378, Mar. 1987.

[47] M. Dal Cin. A Diagnostic Device for Large Multiprocessor Systems. *Proc. of the 12th IEEE Intl. Symp. on Fault-Tolerant Computing*, pages 357–360, June 1982.

[48] S. K. Das, S. R. Ohring, and A. K. Banerjee. Embeddings Into Hyper Petersen Networks: Yet Another Hypercube-Like Interconnection Topology. *VLSI Design*, 2(4):335–351, 1995.

[49] D. Dasgupta, K. KrishnaKumar, D. Wong, and M. Berry. Negative Selection Algorithm for Aircraft Fault Detection. *Proc. of the 3rd Intl. Conf. on Artificial Immune Systems*, pages 1–13, Sept. 2004.

[50] J. Davies. *Implementing SSL / TLS: Using Cryptography and PKI.* Wiley, Jan. 2011.

[51] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media Over a Peer-to-Peer Network. *Technical Report, Stanford InfoLab*, (2001-30), Apr. 2001.

[52] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. *Proc. of the Workshop on Peer-to-peer Streaming and IP-TV*, pages 323–328, Aug. 2007.

[53] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. Pollution in P2P Live Video Streaming. *Intl. Journal of Computer Networks and Communications*, 1(2):99–110, July 2009.

[54] E. P. Duarte Jr., A. Brawerman, and L. C. P. Albini. An Algorithm for Distributed Hierarquical Diagnosis of Dynamic Fault and Repair Events. *Proc. of the IEEE Intl. Conf. on Parallel and Distributed Systems*, pages 299–306, 2000.

[55] E. P. Duarte Jr. and T. Nanya. Multi-Cluster Adaptive Distributed System-Level Diagnosis Algorithms. *IEICE Techinical Report FTS 95-73*, 1995.

[56] E. P. Duarte Jr. and T. Nanya. A Hierarquical Adaptive Distributed System-Level Diagnosis Algotithm. *IEEE Transactions on Computers*, 47(1):34–45, Jan. 1998.

[57] E. P. Duarte Jr. and A. Weber. A Distributed Network Connectivity Algorithm. *Proc. of the 6th IEEE Intl. Symp. on Autonomous Decentralized Systems*, pages 285–292, Apr. 2003.

[58] E. P. Duarte Jr., A. Weber, and K. V. O. Fonseca. Distributed Diagnosis of Dynamic Events in Partitionable Arbitrary Topology Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1415–1426, Aug. 2012.

[59] E. P. Duarte Jr., R. P. Ziwich, and L. C. P. Albini. A Survey of Comparison-Based System-Level Diagnosis. *ACM Computing Surveys*, 43(3):22:1–22:56, Apr. 2011.

[60] K. Efe. A Variation on the Hypercube with Lower Diameter. *IEEE Transactions on Computers*, 40(11):1312–1316, Nov. 1991.

[61] K. Efe. The Crossed Cube Architecture for Parallel Computing. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):513–524, Sept. 1992.

[62] K. Efe, P. K. Blackwell, W. Slough, and T. Shiau. Topological Properties of the Crossed Cubes Architecture. *IEEE Transactions on Computers*, 44(7):923–929, July 1995.

[63] A. El-Amawy and S. Latifi. Properties and Performance of Folded Hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):31–42, Jan. 1991.

[64] M. Elhadef. A Perceptron Neural Network for Asymmetric Comparison-Based System-Level Fault Diagnosis. *Proc of the 5th Intl. Conf. on Availability, Reliability and Security*, pages 265–272, Mar. 2009.

[65] M. Elhadef. A Modified Hopfield Neural Network for Diagnosing Comparison-Based Multiprocessor Systems Using Partial Syndromes. *Proc. of the 17th IEEE Intl. Conf. on Parallel and Distributed Systems*, pages 646–653, Dec. 2011.

[66] M. Elhadef. Using Linear Support Vector Machines to Solve the Asymmetric Comparison-Based Fault Diagnosis Problem. *Proc of the 7th Intl. Conf. on Availability, Reliability and Security*, pages 18–27, Aug. 2012.

[67] M. Elhadef and B. Ayeb. An Evolutionary Algorithm for Identifying faults in t-Diagnosable Systems. *Proc. of the 19th Symp. on Reliable Distributed Systems*, pages 74–83, Oct. 2000.

[68] M. Elhadef and B. Ayeb. Efficient Comparison-Based Fault Diagnosis of Multiprocessor Systems Using an Evolutionary Approach. *Proc. of the 15th Intl. Parallel and Distributed Processing Symp.*, 1:6, Apr. 2001.

[69] M. Elhadef and B. Ayeb. Self-Diagnosis of Multiprocessor Systems Under Generalized Comparison Model. *Proc. of the ISCA Intl. Conf. on Parallel and Distributed Computing Systems*, pages 372–379, Aug. 2001.

[70] M. Elhadef and B. Ayeb. An Evolutionary Algorithm for Generalized Comparison-Based Self-Diagnosis of Multiprocessor Systems. *Applied Artificial Intelligence*, 16(1):73–95, Jan. 2002.

[71] M. Elhadef, A. Boukerche, and H. Elkadiki. Diagnosing Mobile Ad Hoc Networks: Two Distributed Comparison-Based Self-Diagnosis Protocols. *Proc. of the 4th ACM*

Intl. Workshop on Mobility Management and Wireless Access, pages 18–27, Oct. 2006.

[72] M. Elhadef, A. Boukerche, and H. Elkadiki. Performance Analysis of a Distributed Comparison-Based Self-Diagnosis Protocol for Wireless Ad Hoc Networks. *Proc. of the 9th ACM Intl. Symp. on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 165–172, Oct. 2006.

[73] M. Elhadef, A. Boukerche, and H. Elkadiki. Self-Diagnosing Wireless Mesh and Ad Hoc Networks Using an Adaptable Comparison-Based Approach. *Proc. of the 2nd Intl. Conf. Availability, Reliability and Security*, pages 983–990, Apr. 2007.

[74] M. Elhadef, S. Das, and A. Nayak. System-Level Fault Diagnosis Using Comparison Models: An Artificial-Immune-Systems-Based Approach. *Journal of Networks*, 1(5):43–53, Oct. 2006.

[75] M. Elhadef and A. Nayak. Efficient Symmetric Comparison-Based Self-Diagnosis Using Backpropagation Artificial Neural Networks. *Proc. of the IEEE 28th Intl. Performance Computing and Communications Conf.*, pages 264–271, Dec. 2009.

[76] M. Elhadef and A. Nayak. A Novel Generalized-Comparison-Based Self-Diagnosis Algorithm for Multiprocessor and Multicomputer Systems Using a Multilayered Neural Network. *Proc. of the 13th IEEE Intl. Conf. on Computational Science and Engineering*, pages 245–252, Dec. 2010.

[77] M. Elhadef and A. Nayak. Comparison-Based System-Level Fault Diagnosis: A Neural Network Approach. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):1047–1059, June 2012.

[78] A.-H. Esfahanian, L. M. Ni, and B. E. Sagan. The Twisted n-Cube with Application to Multiprocessing. *IEEE Transactions on Computers*, 40(1):88–93, Jan. 1991.

[79] J. Fan. Diagnosability of the Möbius Cubes. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):923–928, Sept. 1998.

[80] J. Fan. Diagnosability of Crossed Cubes. *IEEE Transactions on Computers*, 13(10):1099–1104, Oct. 2002.

[81] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and Whitewashing in Peer-to-Peer Systems. *IEEE Journal on Selected Areas in Communications*, 24(5):1010–1019, May 2006.

[82] C. Feng and B. Li. On Large-Scale Peer-to-Peer Streaming Systems with Network Coding. *Proc. of the 16th ACM Intl. Conf. on Multimedia*, pages 269–278, Oct. 2008.

[83] V. Fodor and G. Dan. Resilience in Live Peer-to-peer Streaming. *IEEE Communications Magazine*, 45(6):116–123, June 2007.

[84] A. D. Friedman. A New Measure of Digital System Diagnosis. *Proc. of the 5th IEEE Fault-Tolerant Computing Symp.*, pages 167–169, June 1975.

[85] C. P. Fuhrman and H. J. Nussbaumer. A New Comparison Model in System-Level Diagnosis. *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 687–690, Aug. 1996.

[86] C. P. Fuhrman and H. J. Nussbaumer. Comparison Diagnosis in Large Multiprocessor Systems. *Proc. of the 5th Asian Test Symp.*, pages 244–249, Nov. 1996.

[87] H. Fujiwara and K. Kinoshita. Connection Assignments for Probabilistically Diagnosable Systems. *IEEE Transactions on Computers*, C-27(3):280–283, Mar. 1978.

[88] D. Fussell, M. Malek, and S. Rangarajan. *Wafer-Scale Testing/Design for Testability*, chapter 9, pages 413–472. Kluwer, 1989.

[89] D. Fussell and S. Rangarajan. Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity. *Proc. of the 19th IEEE Fault-Tolerant Computing Symp.*, pages 560–565, June 1989.

[90] M. Garland and D. B. Kirk. Understanding Throughput-Oriented Architectures. *Communications of the ACM*, 53(11):58–66, Nov. 2010.

[91] G. Gheorghe, R. L. Cigno, and A. Montresor. Security and Privacy Issues in P2P Streaming Systems: A Survey. *Peer-to-Peer Networking and Applications*, 4(2):75–91, June 2011.

[92] D. Gourley and B. Totty. *HTTP: The Definitive Guide*. O'Reilly, Sept. 2002.

[93] V. Hadzilacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems, Distributed Systems*. S. Mullender, ACM Press, C.5, 1993.

[94] W. Haizhou, C. Xingshu, and W. Wenxian. A Measurement Study of Polluting a Large-Scale P2P IPTV System. *China Communications*, 8(2):95–102, Mar. 2011.

[95] S. L. Hakimi and A. T. Amin. Characterization of Connection Assignment of Diagnosable Systems. *IEEE Transactions on Computers*, C-23(1):86–88, 1974.

[96] S. L. Hakimi and K. Nakajima. On Adaptive System Diagnosis. *IEEE Transactions on Computers*, C-33(3):234–240, Mar. 1984.

[97] M. Haridasan and R. van Renesse. Defense Against Intrusion in a Live Streaming Multicast System. *Proc. of the 6th IEEE Intl. Conf. on Peer-to-Peer Computing*, pages 185–192, Sept. 2006.

[98] M. Haridasan and R. van Renesse. SecureStream: An Intrusion-Tolerant Protocol for Live-Streaming Dissemination. *Computer Communications*, 31(3):563–575, Feb. 2008.

[99] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8):1672–1687, Dec. 2007.

[100] X. Hei, Y. Liu, and K. W. Ross. IPTV Over P2P Streaming Networks: The Mesh-Pull Approach. *IEEE Communications Magazine*, 46(2):86–92, Feb. 2008.

[101] M. Hollick, I. Martinovic, T. Krop, and I. Rimac. A Survey on Dependable Routing in Sensor Networks, Ad Hoc Networks, and Cellular Networks. *Proc. of the 30th Euromicro Conf.*, pages 495–502, Sept. 2004.

[102] W.-S. Hong and S.-Y. Hsieh. Strong Diagnosability and Conditional Diagnosability of Augmented Cubes Under the Comparison Diagnosis Model. *IEEE Transactions on Reliability*, 61(1):140–148, Mar. 2012.

[103] S. H. Hosseini, J. G. Kuhl, and S. M. Reddy. A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair. *IEEE Transactions on Computers*, C-33(3):223–233, Mar. 1984.

[104] S.-Y. Hsieh and Y.-S. Chen. Strongly Diagnosable Product Networks Under the Comparison Diagnosis Model. *IEEE Transactions on Computers*, 57(6):721–732, June 2008.

[105] S.-Y. Hsieh and Y.-S. Chen. Strongly Diagnosable Systems Under the Comparison Model. *IEEE Transactions on Computers*, 57(12):1720–1725, Dec. 2008.

[106] S.-Y. Hsieh and C.-Y. Kao. Determining the Conditional Diagnosability of k-Ary n-Cubes Under the MM* Model. *Lecture Notes in Computer Science*, 6796:78–88, June 2011.

[107] S.-Y. Hsieh, C.-Y. Tsai, and C.-A. Chen. Strong Diagnosability and Conditional Diagnosability of Multiprocessor Systems and Folded Hypercubes. *IEEE Transactions on Computers*, PP(99), May 2012.

[108] G.-H. Hsu, D.-F. Chiang, L.-M. Shih, L.-H. Hsu, and J. J. M. Tan. Conditional Diagnosability of Hypercubes Under the Comparison Diagnosis Model. *Journal of Systems Architecture*, 55(2):140–146, Feb. 2009.

[109] G. H. Hsu and J. J. M. Tan. Conditional Diagnosability of the BC Networks Under the Comparison Diagnosis Model. *Proc. of the Intl. Computer Symp.*, 1:269–274, Nov. 2008.

[110] B. Hu and H. Zhao. Joint Pollution Detection and Attacker Identification in Peer-to-Peer Live Streaming. *Proc. of the IEEE Intl. Conf. on Acoustics Speech and Signal Processing*, pages 2318–2321, Mar. 2010.

[111] Internet World Stats. "World Internet Usage Statistics News and World Population Stats". *http://www.internetworldstats.com/stats.htm.* Accessed in Jan 2013.

[112] Y. Ishida. Active Diagnosis by Self-Organization: An Approach by The Immune Network Metaphor. *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pages 1084–1091, Aug. 1997.

[113] P. Jalote. *Fault Tolerance in Distributed Systems.* Prentice Hall, 1994.

[114] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. *Proc. of the 1st ACM SIGOPS/EuroSys European Conf. on Computer Systems*, pages 3–13, Apr. 2006.

[115] A. Kavianpour. Sequential Diagnosability of Star Graphs. *Journal of Computers and Electrical Engineering*, 22(1):37–44, Jan. 1996.

[116] S. W. Keckler and S. K. Reinhardt. Massively Multithreaded Computing Systems. *IEEE Computer*, pages 24–25, Aug. 2012.

[117] W. E. Kozlowski and H. Krawczyk. A Comparison-Based Approach in Multicomputer System Diagnosis in Hybrid Fault Situations. *IEEE Transations on Computers*, 40(11):1283–1286, Nov. 1991.

[118] S. E. Kreutzer and S. L. Hakimi. Adaptive Fault Identification in Two Diagnostic Models. *Proc. of the 21th Allerton Conf. on Communication, Control and Computing*, pages 353–362, Mar. 1983.

[119] J. G. Kuhl. Fault Diagnosis in Computing Networks. *Dep. Elec. Comput. Eng., Univ. of Iowa, Technical Report*, Aug. 1980.

[120] J. G. Kuhl and S. M. Reddy. Distributed Fault-Tolerance for Large Multiprocessor Systems. *Proc. of the 7th Annual Intl. Symp. on Computer Architecture*, pages 23–30, May 1980.

[121] J. G. Kuhl and S. M. Reddy. Fault-Diagnosis in Fully Distributed Systems. *Proc. of the 11th IEEE Fault-Tolerant Computing Symp.*, pages 100–105, June 1981.

[122] P. Kulasinghe and S. Bettayeb. Embedding Binary Trees into Crossed Cubes. *IEEE Transactions on Computers*, 44(7):923–929, July 1995.

[123] L. E. LaForge, K. F. Kover, and M. S. Fadali. What Designers of Bus and Networks Architectures Should Know about Hypercubes. *IEEE Transactions on Computers*, 52(4):525–533, Apr. 2003.

[124] P.-L. Lai, J. J. Tan, C.-H. Tsai, and L.-H. Hsu. The Diagnosability of the Matching Composition Netork Under the Comparison Diagnosis Model. *IEEE Transactions on Computers*, 53(8):1064–1069, Aug. 2004.

[125] P.-L. Lai, J. J. M. Tan, C.-P. Chang, and L.-H. Hsu. Conditional Diagnosability Measures for Large Multiprocessor Systems. *IEEE Transactions on Computers*, 54(2):165–175, Feb. 2005.

[126] L. A. Laranjeira, M. Malek, and R. M. Jenevein. On Tolerating Faults in Naturally Redundant Algorithms. *Proc. of the 10th IEEE Symp. Reliable Distributed Systems*, pages 118–127, Oct. 1991.

[127] C. W. Lee and S. Y. Hsieh. Diagnosability of Two-Matching Composition Networks Under the MM* Model. *IEEE Transactions on Dependable and Secure Computing*, 8(2):246–255, Apr. 2011.

[128] S. Lee and K. G. Shin. On Probabilistic Diagnosis of Multiprocessor Systems Using Multiple Syndromes. *IEEE Transactions on Parallel and Distributed Systems*, 5(6):630–638, June 1994.

[129] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[130] F. T. Leighton, B. M. Maggs, and R. K. Sitaraman. On The Fault Tolerance of Some Popular Bounded-Degree Networks. *SIAM Journal on Computing*, 27(5):1303–1333, Oct. 1998.

[131] H. Li, H. Wang, and G. Feng. Adaptive Hierarchical Intrusion Tolerant Model Based on Autonomic Computing. *Proc. of the Intl. Conf. on Security Technology*, pages 137–141, Dec. 2008.

[132] J.-S. Li, C.-J. Hsieh, and Y.-K. Wang. Distributed Key Management Scheme for Peer-to-Peer Live Streaming Services. *Intl. Journal of Communication Systems*, Feb. 2012.

[133] J. Liang, R. Kumar, and K. W. Ross. The FastTrack Overlay: A Measurement Study. *Computer Networks*, 50(6):842–858, Apr. 2006.

[134] J. Liang, N. Naoumov, and K. W. Ross. Efficient Blacklisting and Pollution-Level Estimation in P2P File-Sharing Systems. *Asian Internet Engineering Conference*, pages 173–175, Dec. 2005.

[135] E. Lin, D. M. N. de Castro, M. Wang, and J. Aycock. SPoIM: A Close Look at Pollution Attacks in P2P Live Streaming. *Proc. of the 18th Intl. Workshop on Quality of Service*, pages 1–9, June 2010.

[136] F. Lombardi. Comparison-Based Diagnosis with Faulty Comparators. *Eletronic Letters*, 22(22):1158–1160, Oct. 1986.

[137] T. Loocher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. *Proc. of the 21st Intl. Symp. on Distributed Computing*, pages 388–402, Sept. 2007.

[138] M. Lu, P. P. C. Lee, and J. C. S. Lui. Identity Attack and Anonymity Protection for P2P-VoD Systems. *Proc. of the ACM/IEEE Intl. Workshop on Quality of Service*, pages 1–9, June 2011.

[139] M. J. Ma and J. M. Xu. Panconnectivity of Locally Twisted Cubes. *Appl. Math. Lett.*, 19(7):673–677, July 2006.

[140] J. Maeng and M. Malek. A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems. *Proc. of the 11th IEEE Fault-Tolerant Computing Symp.*, pages 173–175, Apr. 1981.

[141] P. Maestrini and P. Santi. Self Diagnosis of Processor Arrays Using a Comparison Model. *Proc. of the 14th Symp. on Reliable Distributed Systems*, pages 218–228, Sept. 1995.

[142] S. N. Maheshwari and S. L. Hakimi. On Models for Diagnosable Systems and Probabilistic Fault Diagnosis. *IEEE Transactions on Computers*, C-25(3):228–236, Mar. 1976.

[143] M. Malek. A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems. *Proc. of the 7th Annual Intl. Symp. on Computer Architecture*, pages 31–36, May 1980.

[144] F. S. Martins, R. M. Andrade, A. L. Santos, B. Schulze, and J. N. Souza. Detecting Misbehaving Units on Computational Grids. *Concurrency and Computation: Practice & Experience*, 22(3):329–342, Mar. 2009.

[145] F. S. Martins, R. M. C. Andrade, A. L. Santos, J. N. Souza, and B. Schulze. Diagnosis on Computational Grids for Detecting Intelligent Cheating Nodes. *Proc. of the 2nd Intl. Latin American Grid Workshop*, pages 7–14, Nov. 2008.

[146] F. S. Martins, M. Maia, R. M. Andrade, A. L. Santos, and J. N. de Souza. A Grid Computing Diagnosis Model for Tolerating Manipulation Attacks. *Intl. Transactions on Systems Science and Applications*, 2(2):135–146, 2006.

[147] F. S. Martins, M. Maia, R. M. Andrade, A. L. Santos, and J. N. de Souza. Detecting Malicious Manipulation in Grid Environments. *Proc. of the 18th Intl. Symp. on Computer Architecture and High Performance Computing*, pages 28–35, Oct. 2006.

[148] G. Masson, D. Blough, and G. Sullivan. *System Diagnosis.* Prentice-Hall, 1996.

[149] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ Algorithm for Maximum Matching in General Graphs. *Proc. of the 16th Annual Symp. Foundations of Comput. Science*, pages 17–27, Oct. 1980.

[150] G. Montassier, T. Cholez, G. Doyen, R. Khatoun, I. Chrisment, and O. Festor. Content Pollution Quantification in Large P2P Networks: A Measurement Study on KAD. *IEEE Intl. Conf. on Peer-to-Peer Computing*, pages 30–33, Sept. 2011.

[151] K. Nakajima. A New Approach to System Diagnosis. *Proc. of the 19th Allerton Conf. on Communication, Control and Computing*, pages 697–706, Sept. 1981.

[152] B. T. Nassu, E. P. Duarte Jr., and A. T. R. Pozo. A Comparison of Evolutionary Algorithms for System-Level Diagnosis. *Proc. of the 7th ACM Genetic and Evolutionary Computation Conf.*, pages 2053–2060, June 2005.

[153] J. Oliveira, A. Borges, and S. Campos. Content Pollution on P2P Live Streaming Systems. *Proc. of the 15th Brazilian Symposium on Multimedia and the Web*, Oct. 2009.

[154] N. Oualha and Y. Roudier. A Game Theoretical Approach in Securing P2P Storage Against Whitewashers. *Proc. of the 18th IEEE Intl. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 128–133, July 2009.

[155] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. E. Mohr, and E. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. *Proc. of the 4th Intl. Workshop on Peer-To-Peer Systems*, pages 127–140, Feb. 2005.

[156] A. Pelc. Undirected Graph Models for System-Level Fault Diagnosis. *IEEE Transactions on Computers*, 40(11):1271–1276, Nov. 1991.

[157] A. Pelc. Optimal Fault Diagnosis in Comparison Models. *IEEE Transations on Computers*, 41(6):779–786, June 1992.

[158] F. Preparata, G. Metze, and R. T. Chien. On the Connection Assignment Problem of Diagnosable Systems. *IEEE Transactions on Computers*, 16:848–854, Dec. 1967.

[159] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2004.

[160] V. Raghavan and A. R. Tripathi. Sequential Diagnosability is co-NP-Complete. *IEEE Transactions on Computers*, 40(5):584–595, May 1991.

[161] H. V. Ramasamy, P. Pandey, M. Cukier, and W. H. Sanders. Experiences with Building an Intrusion-Tolerant Group Communication System. *Software - Practice & Experience*, 38(6):639–666, May 2008.

[162] S. Rangarajan, A. T. Dahbura, and E. A. Ziegler. A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies. *IEEE Transactions on Computers*, 44(2):312–333, Feb. 1995.

[163] S. Rangarajan and D. Fussell. A Probabilistic Method for Fault Diagnosis of Multiprocessor Systems. *Proc. of the 18th IEEE Fault-Tolerant Computing Symp.*, pages 278–283, June 1988.

[164] S. Rangarajan, D. Fussell, and M. Malek. Built-in Testing of Integrated Circuits Wafers. *IEEE Transactions on Computers*, 39(2):195–205, Feb. 1990.

[165] R. D. Rettberg. *Shared Memory Parallel Processing: The Butterfly and the Monarch.* MIT Press, 1986.

[166] B. Sallay, P. Maestrini, and P. Santi. Wafer-Scale Diagnosis Tolerating Comparator Faults. *IEE Proc. Computer and Digital Techniques*, 146(4):211–215, July 1999.

[167] E. A. Schimidt, R. P. Ziwich, E. P. Duarte Jr., and I. Jansch-Pôrto. Diagnóstico de Poluição de Conteúdo em Redes P2P para Transmissões de Mídia Contínua ao Vivo. *Proc. of the 17th Brazilian Symposium on Multimedia and the Web*, pages 221–228, Oct. 2011.

[168] J. Seibert, X. Sun, C. Nita-Rotaru, and S. Rao. Towards Securing Data Delivery in Peer-to-Peer Streaming. *Proc. of the 2nd Intl. Conf. on Comunication Systems and Networks*, pages 1–10, Jan. 2010.

[169] A. Sengupta and A. T. Dahbura. On Self-Diagnosable Multiprocessor Systems: Diagnosis by Comparison Approach. *IEEE Transactions on Computers*, 41(11):1386–1396, Nov. 1992.

[170] A. Sengupta and C. Rhee. On the Diagnosability of Systems with Three Valued Test Results: Diagnosis by Comparison Strategy. *Proc. of the 20th Intl. Symp. on Multiple-Valued Logic*, pages 115–120, May 1990.

[171] J.-J. Sheu, W.-T. Huang, and C.-H. Chen. Strong Diagnosability of Regular Networks Under the Comparison Model. *Information Processing Letters*, 106(1):19–25, Mar. 2008.

[172] J. So and D. Reeves. AntiLiar: Defending Against Cheating Attacks in Mesh Based Streaming. *Proc. of the IEEE 12th Intl. Conf. on Peer-to-Peer Computing*, pages 115–125, Sept. 2012.

[173] M. Stahl, R. Buskens, and R. Bianchini. Simulation of the Adapt On-Line Diagnosis Algorithm for General Topology Networks. *Proc. of the 11th IEEE Symp. Reliable Distributed Systems*, pages 180–187, Oct. 1992.

[174] I. A. Stewart. A General Algorithm for Detecting Faults Under the Comparison Diagnosis Model. *Proc. of the 24th IEEE Intl. Symp. on Parallel and Distributed Processing*, pages 1–9, Apr. 2010.

[175] J. A. Stratton, C. Rodrigues, I.-J. Sung, L.-W. Chang, N. Anssari, G. Liu, and W.-M. W. Hwu. Algorithm and Data Optimization Techniques for Scaling to Massively Threaded Systems. *IEEE Computer*, pages 26–32, Aug. 2012.

[176] A. Subbiah and D. M. Blough. Distributed Diagnosis in Dynamic Fault Environments. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):453–467, May 2004.

[177] G. Sullivan. An $O(t^3 + |E|)$ Fault Identification Algorithm for Diagnosable Systems. *IEEE Transactions on Computers*, 37(4):388–397, Apr. 1988.

[178] H. Tamaki. Efficient Self-Embedding of Butterfly Networks with Random Faults. *SIAM Journal on Computing*, 27(3):614–636, June 1998.

[179] N. F. Tzeng and S. Wei. Enhanced Hypercubes. *IEEE Transactions on Computers*, 40(3):284–294, Mar. 1991.

[180] US-CERT. "United States Computer Emergency Readiness Team". *http://www.us-cert.gov*. Accessed in Dec. 2012.

[181] A. S. Vaidya, P. S. N. Rao, and S. R. Shankar. A Class of Hypercube-like Networks. *Proc. of the 5th IEEE Symp. Parallel and Distributed Processing*, 1(4):800–803, Dec. 1993.

[182] V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1998.

[183] A. Vieira, S. Campos, and J. Almeida. Fighting Attacks in P2P Live Streaming. Simpler is Better. *Proc. of the 28th IEEE Intl. Conf. on Computer Communications Workshops*, pages 355–356, Apr. 2009.

[184] A. B. Vieira. Transmissão de Mídia Contínua ao Vivo em P2P: Modelagem, Caracterização e Implementação de Mecanismos de Resiliência a Ataques. *Tese de Doutorado, Universidade Federal de Minas Gerais (UFMG)*, 2010.

[185] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. *Proc. of the 3rd USENIX Symp. on Networked Systems Design and Implementation*, 3:1–14, May 2006.

[186] D. Wang. Diagnosability of Hipercubes and Enhanced Hypercubes Under the Comparison Diagnosis Model. *IEEE Transactions on Computers*, 48(12):1369–1374, Dec. 1999.

[187] H. Wang, D. M. Blough, and L. Alkalaj. Analysis and Experimental Evaluation of Comparison-Based System-Level Diagnosis for Multiprocessor Systems. *Proc. of the 24th IEEE Fault-Tolerant Computing Symp.*, pages 55–64, June 1994.

[188] H. Wang, D. M. Blough, and L. Alkalaj. Practical Approach to Comparison-based Fault Diagnosis in Multiprocessor Systems. *Intl. Journal of Computer Systems Science and Engineering*, 9(1):11–20, Jan. 1994.

[189] M. Wang and B. Li. Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming. *Proc. of the 26th IEEE Intl. Conf. on Computer Communications*, pages 1082–1090, May 2007.

[190] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana. MIS: Malicious Nodes Identification Scheme in Network-Coding-Based Peer-to-Peer Streaming. *Proc. of the 29th IEEE Intl. Conf. on Computer Communications*, pages 1–5, Mar. 2010.

[191] C. K. Wong and S. S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, Aug. 1999.

[192] J. Xu and S. Huang. A New Comparison-Based Scheme for Multiprocessor Fault Tolerance. *Microprocessing and Microprogramming*, 30(1–5):617–623, Aug. 1990.

[193] J. Xu and B. Randell. Software Fault Tolerance: $t/(n-1)$-Variant Programming. *IEEE Transactions on Reliability*, 46(1):60–68, Mar. 1997.

[194] C.-L. Yang and G. M. Masson. An Efficient Algorithm for Multiprocessor Fault Diagnosis Using the Comparison Approach. *Information and Computation*, 74(1):50–63, July 1987.

[195] H. Yang and X. Yang. A Fast Diagnosis Algorithm for Locally Twisted Cube Multiprocessor Systems under the MM* Model. *Computers & Mathematics with Applications*, 53(6):918–926, Mar. 2007.

[196] S. Yang, H. Jin, B. Li, X. Liao, H. Yao, and X. Tu. The Content Pollution in Peer-to-Peer Live Streaming Systems: Analysis and Implications. *Proc. of the 37th Intl. Conf. on Parallel Processing*, pages 652–659, Sept. 2008.

[197] X. Yang. A Linear Time Fault Diagnosis Algorithm for Hypercube Multiprocessors Under the MM* Model. *Proc. of the 12th Asian Test Symp.*, pages 50–55, Nov. 2003.

[198] X. Yang and Y. Y. Tang. Efficient Fault Identification of Diagnosable Systems Under the Comparison Model. *IEEE Transactions on Computers*, 56(12):1612–1618, Dec. 2007.

[199] X. F. Yang, D. J. Evans, and G. M. Megson. Locally Twisted Cubes are 4-Pancyclic. *Appl. Math. Lett.*, 17(8):919–925, Aug. 2004.

[200] X. F. Yang, D. J. Evans, and G. M. Megson. The Locally Twisted Cubes. *Intl. Journal of Computer Mathematics*, 82(4):401–413, Apr. 2005.

[201] X. F. Yang, G. M. Megson, and D. J. Evans. A Comparison-Based Diagnosis Algorithm Tailored for Crossed Cube Multiprocessor Systems. *Microprocessors and Microsystems*, 19(4):169–175, May 2005.

[202] X. Yu and S. Fujita. Whitewash-Aware Reputation Management in Peer-to-Peer File Sharing System. *Proc. of the World Congress in Computer Science, Computer Engineering, and Applied Computing*, July 2012.

[203] P. Zhang and B. E. Helvik. Modeling and Analysis of P2P Content Distribution Under Coordinated Attack Strategies. *IEEE Consumer Communications and Networking Conf.*, pages 131–135, Jan. 2011.

[204] J. Zheng, S. Latifi, E. Regentova, K. Luo, and X. Wu. Diagnosability of Star Graphs under the Comparison Diagnosis Model. *Information Processing Letters*, 16(1):73–95, Jan. 2002.

[205] S. Zhou. The Conditional Diagnosability of Crossed Cubes Under the Comparison Model. *Intl. Journal of Computer Mathematics*, 87(15):3387–3396, Dec. 2010.

[206] Q. Zhu. On Conditional Diagnosability and Reliability of the BC Networks. *The Journal of Supercomputing*, 45(2):173–184, Aug. 2008.

[207] Q. Zhu, X.-K. Wang, and G. Cheng. Reliability Evaluation of BC Networks. *IEEE Transactions on Computers*, PP(99):1–6, 2012.

[208] R. P. Ziwich, E. P. Duarte Jr., and L. C. P. Albini. Distributed Integrity Checking for System with Replicated Data. *Proc. of the 11th IEEE Intl. Conf. on Parallel and Distributed Systems*, pages 363–369, July 2005.

[209] R. P. Ziwich, E. A. Schimidt, E. P. Duarte Jr., and I. Jansch-Pôrto. Diagnosis of Content Pollution in P2P Live Streaming Networks. *Proc. of the 6th Latin-American Symp. on Dependable Computing*, pages 48–57, Apr. 2013.

APÊNDICE A

OUTRAS ABORDAGENS PARA O DIAGNÓSTICO BASEADO EM COMPARAÇÕES

Uma série de outras abordagens para o diagnóstico em nível de sistema baseado em comparações foi apresentada nas últimas 3 décadas. As abordagens incluem diferentes modelos de diagnóstico baseado em comparações, algoritmos e análises dos limites de diagnosticabilidade para diversas topologias. Este apêndice apresenta um *survey* – baseado em [59], além de trabalhos mais recentes – de diversas outras abordagens para o diagnóstico baseado em comparações, diferentes das apresentadas no Capítulo 2.

Na sequência, este apêndice está dividido em 14 seções e apresenta, nesta ordem: o diagnóstico e a diagnosticabilidade de hipercubos e *enhanced hypercubes*, redes borboletas, cubos cruzados, *locally twisted cubes* e *hypercube-like networks*, grafos estrela, *matching composition networks*, redes t-conectadas e redes produto; também são apresentados os resultados recentes da diagnosticabilidade forte e da diagnosticabilidade condicional, o modelo de comparações baseado em *broadcast*, as abordagens probabilística e também as evolucionária baseadas em comparações, e os modelos baseados em comparações para redes ad hoc; por fim, a última seção apresenta um sumário dos resultados relevantes do diagnóstico em nível de sistema baseado em comparações. Este sumário inclui grafos cronologicamente ordenados que mostram o relacionamento entre os diversos resultados do diagnóstico em nível de sistema baseado em comparações, e também tabelas com um resumo mais detalhado de todos os resultados.

A.1 Diagnóstico Baseado em Comparações para Hipercubos

O hipercubo (*hypercube*) é uma topologia escalável para conexão entre nodos de um sistema [123]. Várias propriedades dos hipercubos permitem que características como alta performance e tolerância a falhas sejam facilmente incorporadas ao sistema. A diagnosticabilidade de hipercubos e dos "hipercubos melhorados" (*enhanced hypercubes*) [179] no modelo MM* foi proposta por [186]. A topologia do sistema é representada por um grafo $G = (V, E)$, onde cada nodo $i \in V$ representa os nodos do sistema e cada aresta $(i, j) \in E$ representa um enlace (ou *link*) de comunicação entre os nodos i e j. As comparações executadas no sistema são modeladas através de um multigrafo $M = (V, C)$. V representa o conjunto dos nodos do sistema, e uma aresta $(i, j)_k \in C$, onde k é o rótulo da aresta que conecta i e j, indica que o nodo i e o nodo j são comparados pelo nodo k.

Um hipercubo n-dimensional – também chamado de um n-hipercubo ou ainda H_n – pode ser visto como um grafo $G = (V, E)$ onde V consiste de 2^n nodos, nomeados de $00\ldots0$ até $11\ldots1$ (n *bits*). Uma aresta $(i, j) \in E$ se e somente se i e j tem somente um *bit* diferente. Por este motivo, todos os nodos estão conectados a exatos outros n nodos. Se dois nodos i e j de um n-hipercubo possuem d *bits* diferentes, é dito que estes dois nodos possuem *distância de Hamming* (H) igual a d, denotado por $H(i, j) = d$. Então, em um n-hipercubo existe uma conexão entre i e j se e somente se $H(i, j) = 1$. Como um exemplo, a Figura A.1(a) mostra um 3-hipercubo e a Figura A.1(b) mostra um 4-hipercubo.

Em [179] os *enhanced hypercubes* são definidos através da adição de *links* extras de conexões – também conhecidos como *desvios* (*detours*) – aos hipercubos regulares. Estas estruturas diminuem a distância entre nodos e o diâmetro do sistema, entre outras características. Um *enhanced hypercube* é denotado por (n, k)-hipercubo e é construído através da adição de 2^{n-1} *links* extras ao n-hipercubo correspondente. Existe um desvio entre um par de nodos com rótulos $b_n b_{n-1} \ldots b_{k+1} b_k b_{k-1} \ldots b_1$ e $b_n b_{n-1} \ldots b_{k+1} \bar{b}_k \bar{b}_{k-1} \ldots \bar{b}_1$ onde $\bar{b}_i$ é o complemento de b_i, e $k \in \{2, \ldots, n\}$ é a distância de Hamming entre os pares de

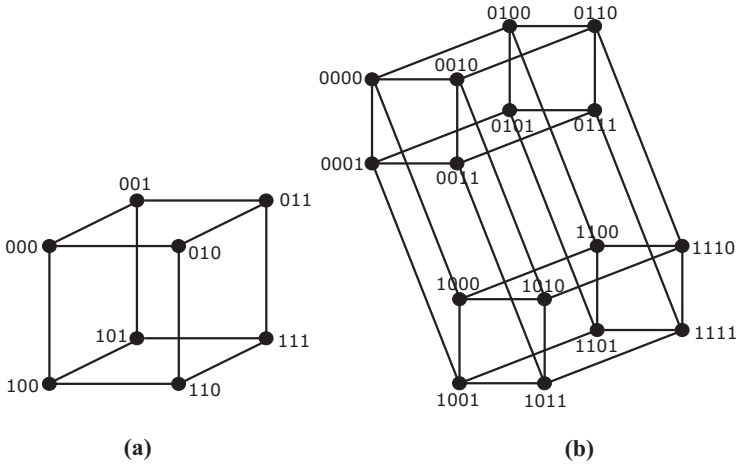


Figura A.1: (a) Um 3-hipercubo. (b) Um 4-hipercubo.

nodos conectados pelo desvio. Exemplos de um $(3, 2)$-hipercubo e de um $(3, 3)$-hipercubo são mostrados nas Figuras A.2(a) e A.2(b), respectivamente. Nestas figuras as linhas pontilhadas correspondem aos desvios dos *enhanced hypercubes*.

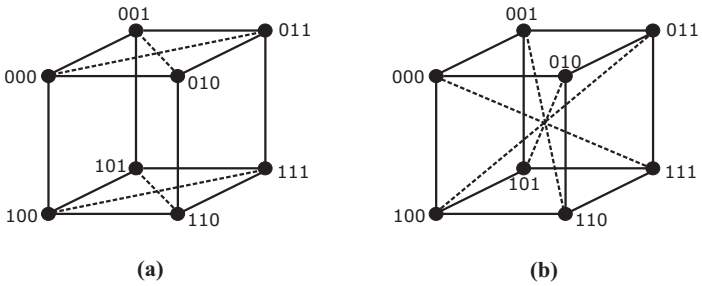


Figura A.2: (a) Um $(3, 2)$-hipercubo. (b) Um $(3, 3)$-hipercubo. Os desvios aparecem em linhas pontilhadas.

A diagnosticabilidade de n-hipercubos é provada [186], e é n sobre o modelo MM*, se $n \geq 5$, considerando um sistema com $N = 2^n$ nodos. A diagnosticabilidade de *enhanced hypercubes* é $n + 1$ sobre o mesmo modelo de diagnóstico, se $n \geq 6$.

Wang [186] primeiramente define uma cobertura de vértices, que é um subconjunto $K \subseteq V$ tal que toda aresta de E é adjacente a um nodo em K. A ordem do vértice i é então definida como a cardinalidade do menor subgrafo de cobertura G_i, construído com o subconjunto de nodos que são comparados com i com as arestas das comparações correspondentes.

A prova é baseada na caracterização previamente apresentada por Sengupta e Dahbura

[169] (descrita na Seção 2.3) que apresenta o conjunto de condições que garantem um sistema ser t-diagnosticável:

1. $N \geq 2t + 1$, e
2. cada nodo possui ordem pelo menos t, e
3. para cada $V' \subset V$, tal que $|V'| = N - 2t + p$ e $0 \leq p \leq t - 1$, o número de nodos que não estão contidos em V' mas que são comparados com algum nodo de V' e por algum nodo de V' é maior que p.

A condição 1, $2^n \geq 2n + 1$ é trivialmente verdadeira quando $n \geq 3$. Esta condição é válida para ambos os hipercubos e *enhanced hypercubes*. A condição 2, é satisfeita pela prova de que todo nodo de um n-hipercubo possui ordem n em um hipercubo e ordem $n + 1$ em um *enhanced hypercube*, então em ambos os casos a ordem de um nodo é maior que t. Finalmente Wang mostra que um 5-hipercubo é o menor hipercubo e que os $(6, k)$-hipercubos são os menores *enhanced hypercubes* que satisfazem a terceira das condições apresentadas por Sengupta e Dahbura.

Em ambos os casos, após conhecida a diagnosticabilidade de hipercubos e *enhanced hypercubes*, é possível aplicar o algoritmo de diagnóstico proposto neste trabalho, no Capítulo 3, para encontrar os nodos falhos do sistema. Além disso, também é possível aplicar o algoritmo de diagnóstico $O(N^5)$ proposto em [169] ou o algoritmo $O(N\Delta^3\delta)$ proposto em [198]. Yang em [197] também apresenta um algoritmo de diagnóstico baseado em comparações específico para hipercubos n-dimensionais onde $n \geq 9$, que possui ordem de complexidade $O(Nlog_2^2 N)$ no pior caso.

A.2 Diagnóstico Baseado em Comparações para Redes Borboletas

A rede *borboleta* (*butterfly*) [165, 129] é outra topologia para interconexão de redes que possui vantagens para computação tolerante a falhas [130, 178]. A diagnosticabilidade

de redes borboleta sobre a abordagem baseada em comparações é apresentada por Araki e Shibata [10]. Este trabalho também é baseado no modelo de diagnóstico baseado em comparações apresentado por Maeng e Malek, e sua motivação também é o fato mostrado por Sengupta e Dahbura onde indicam ser custoso calcular a diagnosticabilidade de redes de topologias arbitrárias, portanto deve ser calculada caso a caso.

A rede borboleta, denotadas por $BF(k, r)$ – e também chamada de "borboleta embrulhada" (*wrapped butterfly*) [129] – é uma borboleta k-aria r-dimensional e possui rk^r nodos. Cada nodo tem um rótulo $\langle \ell; x_0 x_1 \dots x_{r-1} \rangle$, onde $0 \leq \ell \leq r-1$, $0 \leq x_i \leq k-1$, e $0 \leq i \leq r-1$. O símbolo ℓ nos rótulos representa o nível (*level*) dos nodos. O nível ℓ indica a coluna dos nodos na representação da topologia borboleta.

Cada nodo $\langle \ell; x_0 x_1 \dots x_{r-1} \rangle$ é adjacente a

$\langle \ell + 1; x_0 \dots x_{\ell-1} y_\ell x_{\ell+1} \dots x_{r-1} \rangle$ para $0 \leq y_\ell \leq k-1$, e

$\langle \ell - 1; x_0 \dots x_{\ell-2} y_{\ell-1} x_\ell \dots x_{r-1} \rangle$ para $0 \leq y_{\ell-1} \leq k-1$.

Como exemplos, uma $BF(2, 3)$ e uma $BF(3, 3)$ são mostradas respectivamente na Figura A.3 e na Figura A.4. Nestas figuras os nodos no nível 0 são replicados na 4^a coluna apenas com o propósito de melhorar a visualização.

Em [10] os autores propuseram três esquemas para a configuração de testes em redes borboletas. O primeiro é chamado comparações *one-way*, um esquema de comparações *two-way* é então definido, que finalmente é melhorado em um terceiro esquema: comparações *two-way* melhorada (*enhanced two-way comparison*, ou ETWC). Os três esquemas são descritos a seguir.

No esquema de comparações *one-way*, cada nodo u do nível ℓ compara todo vizinho no nível $\ell + 1$ em pares. Como exemplo, na Figura A.3, o nodo b compara $(a, d)_b$ e o nodo c compara $(a, d)_c$. Como outro exemplo, na Figura A.4 o nodo b compara: $(a, e)_b$, $(a, f)_b$ e $(e, f)_b$. Cada nodo executa $k(k-1)/2$ comparações no esquema *one-way*.

No esquema de comparações *two-way*, cada nodo u do nível ℓ compara todo par de vizinhos do nível $\ell - 1$ e também compara todo par de vizinhos do nível $\ell + 1$. Na Figura

A.3, é possível notar que o nodo a executa duas comparações: $(b, c)_a$ e $(c, f)_a$. Na Figura A.4, um nodo a executa seis comparações: $(b, c)_a$, $(b, d)_a$, $(c, d)_a$, $(x, y)_a$, $(x, z)_a$ e $(y, z)_a$. Cada nodo executa $k(k-1)$ comparações neste esquema.

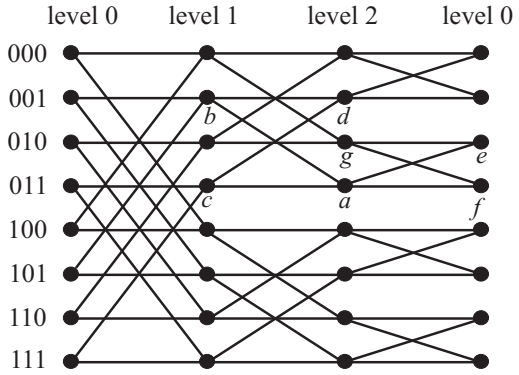


Figura A.3: Uma borboleta $BF(2, 3)$ onde os nodos do nível 0 são replicados.

Araki e Shibata mostram que a diagnosticabilidade de uma rede borboleta $BF(k, r)$ – na qual o esquema de comparações *one-way* é empregado, é $k-2$ para $k \geq 3$ e $r \geq 3$. Considere o exemplo da Figura A.4, nesta borboleta $BF(3, 3)$ existe um total de 81 nodos, mas o diagnóstico só pode ser realizado se no máximo uma unidade é falha. Araki e Shibata mostram que o esquema de comparações *two-way* melhora a diagnosticabilidade destas redes para $2(k-2)$. Para o mesmo exemplo, a diagnosticabilidade é 2. Os autores então mostram que a diagnosticabilidade de borboletas é no máximo $2k$ e propõem outro esquema de comparações que chega neste limite para $k \geq 2$ e $r \geq 5$, o esquema de comparações *two-way* melhorado. Ainda para o mesmo exemplo, agora até 6 nodos podem ser falhos. Considere que para um nodo $u = \langle \ell; x \rangle$, ($x$ é uma *string* k-aria de r-bits), $N^+(U) = \{x_0, x_1, \ldots, x_{k-1}\}$ é o conjunto de k nodos adjacentes ao u no nível $\ell + 1$ e $N^-(U) = \{y_0, y_1, \ldots, y_{k-1}\}$ é o conjunto de k nodos adjacentes ao u no nível $\ell - 1$.

Um nodo u executando o ETWC realiza as seguintes comparações:

1. compara todo par de nodos em $N^+(U)$,
2. compara todo par de nodos em $N^-(U)$, e
3. compara x_i e y_i para cada $0 \leq i \leq k-1$.

Sobre o esquema ETWC, cada nodo realiza k^2 comparações. Como exemplo, na Figura A.4 um nodo a executa as seguintes nove comparações: $(b,c)_a$, $(b,d)_a$, $(c,d)_a$, $(x,y)_a$, $(x,z)_a$, $(y,z)_a$, $(b,x)_a$, $(c,y)_a$ e $(d,z)_a$.

Em outro trabalho [11] os autores propõem um algoritmo de diagnóstico com complexidade $O(k^2n)$ para realizar a localização de falhas em uma $BF(k,r)$. Além disso, também pode-se aplicar o algoritmo $O(N^5)$ para sistemas de topologia arbitrária proposto por Sengupta e Dahbura ou o algoritmo $O(N\Delta^3\delta)$ proposto em [198], além do algoritmo proposto neste trabalho, no Capítulo 3.

A.3 Diagnóstico Baseado em Comparações para Cubos Cruzados

O modelo proposto por [80] avalia a diagnosticabilidade de cubos cruzados (*crossed cubes*) sobre o diagnóstico baseado em comparações. Os cubos cruzados são uma importante variação dos hipercubos [60, 61, 62]. Ambos os cubos cruzados e hipercubos são grafos regulares que possuem o mesmo número de nodos, número de arestas e conectividade; e ambos são recursivos por natureza. Mas o diâmetro de um cubo cruzado é aproximadamente a metade do diâmetro do hipercubo correspondente [60, 30]. Um cubo cruzado com n dimensões contém uma árvore binária completa com $2^n - 1$ nodos e todos os ciclos de tamanho a partir de 4 até $2^n (n \geq 2)$; por outro lado, o hipercubo correspondente não possui estas duas propriedades [122, 30].

O identificador de um nodo x em um cubo cruzado n-dimensional é uma *string* binária de tamanho n e é denotado por $x_{n-1}x_{n-2}\ldots x_0$. O cubo cruzado n-dimensional, também chamado de CQ_n, é um grafo n-regular com $N = 2^n$ nodos e $n2^{n-1}$ arestas. Duas strings binárias $x = x_1x_0$ e $y = y_1y_0$ são par-relacionadas, denotado por $x \sim y$, se e somente se $(x,y) \in \{(00,00),(10,10),(01,11),(11,01)\}$; se x e y não são par-relacionadas, então denota-se $x \not\sim y$.

Um CQ_n é definido recursivamente como apresentado em [60, 61]. CQ_1 é um grafo

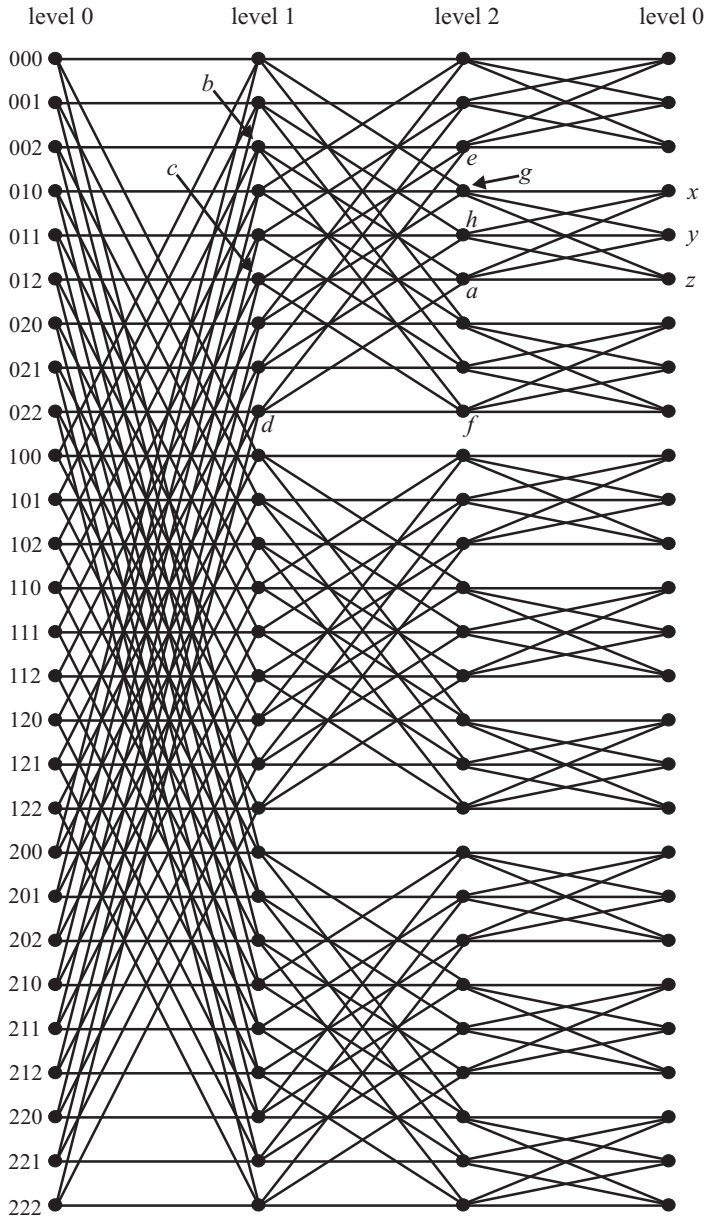


Figura A.4: A estrutura de uma $BF(3,3)$ onde os nodos no nível 0 são replicados.

completo com dois nodos rotulados com 0 e 1, respectivamente. Para $n > 1$, CQ_n consiste de dois sub-cubos CQ^0_{n-1} e CQ^1_{n-1}. O nodo $u = 0u_{n-2} \ldots u_0$ do CQ^0_{n-1} e o nodo $v = 1v_{n-2} \ldots v_0$ do CQ^1_{n-1} são adjacentes, se e somente se:

1. $u_{n-2} = v_{n-2}$ se n for par, e
2. $u_{2i+1}u_{2i} \sim v_{2i+1}v_{2i}$, para $0 \leq i < \lfloor \frac{n-1}{2} \rfloor$.

Como exemplo, a Figura A.5 mostra um cubo cruzado 3-dimensional CQ_3.

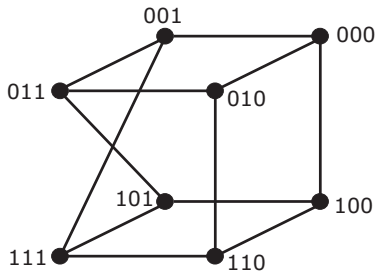


Figura A.5: Um cubo cruzado 3-dimensional CQ_3.

Fan [80] mostra que um cubo cruzado com $n \geq 4$ satisfaz as seguintes condições apresentadas por Sengupta e Dahbura para um sistema ser t-diagnosticável:

1. $2^n \geq 2n + 1$, e
2. cada nodo possui grau no máximo n, e
3. se $n \geq 4$, então para cada $V' \subset V(CQ_n)$, tal que $|V'| = 2^n - 2n + p$ para $0 \geq p \geq n - 1$, o número de nodos que não estão contidos em V' mas que são comparados com algum nodo de V' e por algum nodo de V' é maior que p, onde $V(CQ_n)$ representa o conjunto de vértices do CQ_n.

Fan também prova que os cubos cruzados com $n = 4$ são os menores que satisfazem a estas condições, mostrando que o CQ_3 não satisfaz a terceira condição, enquanto o CQ_1 e o CQ_2 não satisfazem a segunda condição. Fan conclui que a diagnosticabilidade de cubos cruzados n-dimensionais é a mesma dos hipercubos n-dimensionais, isto é, para todo $n \geq 5$, os cubos cruzados são n-diagnosticáveis. Além disso, para $n = 4$, Fan também mostra que a diagnosticabilidade do CQ_4 é 4, enquanto a diagnosticabilidade de um hipercubo 4-dimensional não é 4.

Tanto o algoritmo polinomial apresentado em [169] quanto o algoritmo apresentado em [198], além do proposto no Capítulo 3, podem ser usados para diagnosticar cubos cruzados n-dimensionais se o número de nodos falhos não for maior que n. Além disso Yang, Megson e Evans em [201] apresentam um algoritmo de diagnóstico baseado em comparações específico para cubos cruzados com $n \geq 11$. Este algoritmo possui ordem de complexidade $O(Nlog_2^2 N)$.

A.4 Diagnóstico Baseado em Comparações para *Locally Twisted Cubes* e *Hypercube-Like Multiprocessor Systems*

Yang e Yang em [195] aplicaram o diagnóstico baseado em comparações para sistemas de multiprocessadores baseados em *locally twisted cubes*. Um *locally twisted cube* n-dimensional LTQ_n [200] é uma variante do hipercubo que possui o mesmo número de nodos e arestas como um cubo n-dimensional, mas possui diâmetro menor, além de outras vantagens quando comparado com um hipercubo do mesmo tamanho [200, 199, 139].

Um LTQ_n é definido recursivamente como segue [195, 200]:

1. LTQ_2 é um grafo composto de quatro nodos rotulados com: 00, 01, 10, e 11; respectivamente conectados por quatro arestas: $(00, 01)$, $(01, 11)$, $(11, 10)$, e $(10, 00)$.
2. Para $n \geq 3$, LTQ_n é construído através de duas cópias distintas de LTQ_{n-1} de acordo com os seguintes passos:
   (a) Seja $0LTQ_{n-1}$ um grafo obtido através da cópia do LTQ_{n-1} prefixando o rótulo de cada nodo com 0;
   (b) Seja $1LTQ_{n-1}$ um grafo obtido através da cópia do LTQ_{n-1} prefixando o rótulo de cada nodo com 1;
   (c) Conecte cada nodo $0x_2x_3 \ldots x_n$ do $0LTQ_{n-1}$ ao nodo $1(x_2 \oplus x_n)x_3 \ldots x_n$ do $1LTQ_{n-1}$ com uma aresta, onde $\oplus$ representa a operação binária *xor*.

Como exemplo, a Figura A.6 (a) mostra um *locally twisted cube* 3-dimensional LTQ_3 e (b) mostra um *locally twisted cube* 4-dimensional LTQ_4.

Yang e Yang apresentam um algoritmo de diagnóstico baseado em comparações para sistemas baseados no *locally twisted cube* com base no modelo MM*. O algoritmo pode executar em $O(Nlog_2^2N)$ se estruturas de dados apropriadas forem empregadas [195].

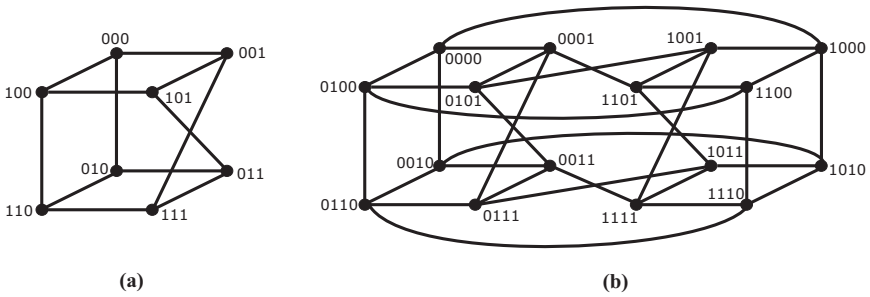


Figura A.6: (a) Um cubo LTQ_3; (b) Um cubo LTQ_4.

Chiang e Tan em [35, 36] aplicam o diagnóstico baseado em comparações para outras topologias *hypercube-like*. Esta classe de interconexão de redes, também chamada de grafos *hypercube-like* (HL), foi primeiramente introduzida por [181]. Grafos HL incluem o hipercubo clássico e muitas outras variantes conhecidas do hipercubo, como o *twisted cube* [78], e o *multi-twisted cube* [60].

Uma rede *hypercube-like* n-dimensional, HL_n, pode ser definida recursivamente de acordo como segue. Considere $V(G_x)$ e $E(G_x)$ como sendo respectivamente o conjunto de vértices e de arestas do grafo G_x. HL_0 é o grafo com um nodo rotulado por 0. Para $n \geq 1$, HL_n consiste de dois HL_{n-1} representados pelos grafos G_0 e G_1, isto é, $HL_n = \{G_0 \cup G_1 \mid G_0, G_1 \text{ são } HL_{n-1}\}$. HL_n possui o conjunto de nodos $V(G_0 \cup G_1) = V(G_0) \cup V(G_1)$ e o conjunto de arestas $E(G_0 \cup G_1) = E(G_0) \cup E(G_1) \cup E_M$, onde E_M é uma arbitrária e perfeita correspondência (*matching*) entre o conjunto de nodos de G_0 e de G_1 em uma forma um-para-um.

A Figura A.7 (c) mostra um exemplo de um HL_3 composto por dois HL_2 mostrados nas Figuras A.7 (a) e (b).

Chiang e Tan [35, 36] provam que a diagnosticabilidade de uma rede *hypercube-like* n-dimensional HL_n é n para $n \geq 5$. Para provar quando um sistema é t-diagnosticável, os autores introduzem um novo conceito chamado diagnosticabilidade local (ou diagnosticabilidade no nodo – *node diagnosability*), que é definida como segue. Um sistema $G = (V, E)$ é t-diagnosticável localmente $x \in V(G)$ se, para cada par de conjuntos distintos $F_1, F_2 \in V(G)$ tal que $|F_1|, |F_2| \leq t$, $F_1 \neq F_2$, e $x \in (F_1 - F_2) \cup (F_2 - F_1)$, o par

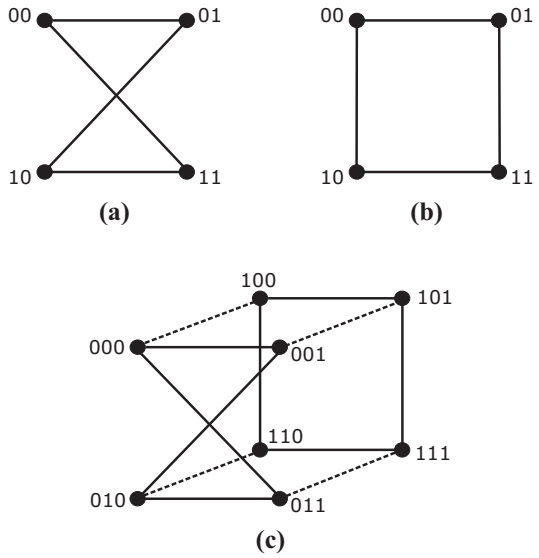


Figura A.7: (a) e (b) Exemplos de HL_2. (c) Um exemplo de um HL_3.

(F_1, F_2) é distinguível. Isto é provado usando a caracterização apresentada por Sengupta e Dahbura [169]. Para todo par de subconjuntos distintos de nodos F_1 e F_2, (F_1, F_2) é um par distinguível se pelo menos uma das seguintes três condições forem satisfeitas:

1. $\exists i, k \in V - F_1 - F_2$ e $\exists j \in (F_1 - F_2) \cup (F_2 - F_1)$ tal que $(i, j)_k \in C$
2. $\exists i, k \in F_1 - F_2$ e $\exists k \in (V - F_1 - F_2)$ tal que $(i, j)_k \in C$
3. $\exists i, k \in F_2 - F_1$ e $\exists k \in (V - F_2 - F_1)$ tal que $(i, j)_k \in C$

Seguindo esta definição, os autores mostram que a diagnosticabilidade local $t_l(x)$ de um nodo $x \in V(G)$ em um sistema $G = (V, E)$ é o número máximo de t para que G seja localmente t-diagnosticável em x, ou seja,

$$t_l(x) = max\{t \mid G \text{ seja localmente } t\text{-diagnosticável em } x\}.$$

Os autores mostram que existe uma relação entre a t-diagnosticabilidade local no nodo x e a t-diagnosticabilidade tradicional, e é apresentada da seguinte forma: um sistema $G = (V, E)$ é t-diagnosticável se e somente se G é localmente t-diagnosticável em x, para $x \in V(G)$. Além disso, os autores provam que um sistema é t-diagnosticável se e somente se $min\{t_l(x) \mid \forall x \in V(G)\} = t$.

Recentemente em [37] Chiang e Tan definiram uma estrutura chamada de estrela estendida para a qual é calculada de forma eficiente a sua diagnosticabilidade local considerando o modelo MM*. Uma estrela estendida, denotada por $ES(x;n)$ de ordem n no nodo x, é definida como segue. Seja x um nodo em um grafo $G = (V, E)$. $ES(x;n) = (V(x;n), E(x;n))$, onde o conjunto de nodos $V(x;n) = \{x\} \cup \{v_{ij} \in V \mid 1 \leq i \leq n, 1 \leq j \leq 4\}$, e o conjunto de arestas $E(x;n) = \{(x, v_{k1}), (v_{k1}, v_{k2}), (v_{k2}, v_{k3}), (v_{k3}, v_{k4}) \mid 1 \leq k \leq n\}$. Em outras palavras, em uma estrela estendida de ordem n no nodo x $ES(x;n)$, existem n caminhos de tamanho 4, com nodos disjuntos, a partir do nodo x. Um exemplo que mostra um nodo x conectado em uma estrutura de estrela estendida é apresentado na Figura A.8.

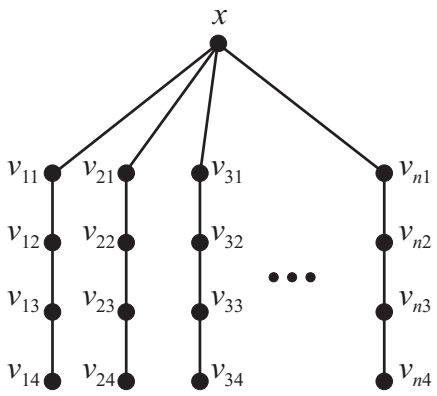


Figura A.8: Uma estrela estendida $ES(x;n)$ no nodo x.

Os autores provam que a diagnosticabilidade local de um nodo x é pelo menos n se existe uma estrela estendida $ES(x;n) \subseteq G$ construída a partir do nodo x. Os autores apresentam um algoritmo para diagnosticar o sistema se existe uma estrutura de estrela estendida em cada nodo. O algoritmo possui ordem de complexidade $O(N\Delta)$, onde N é a quantidade de nodos no sistema e Δ é o grau do nodo de maior grau no sistema.

Por fim, os autores de [174] apresentam um algoritmo para o diagnóstico de falhas em casos especiais de sistemas multiprocessados e distribuídos, com base no modelo MM. O algoritmo proposto possui complexidade $O(\Delta N)$ e as topologias de interconexão de redes que o algoritmo abrange incluem os hipercubos, *enhanced hypercubes*, cubos cruzados, *twisted cubes*, grafos estrela, entre outros.

A.5 Diagnóstico Baseado em Comparações para Grafos Estrela

O grafo estrela (*star graph*) é outra topologia para interconexão de redes que tem sido usada para construir sistemas de múltiplos computadores tolerantes a falhas [115]. A diagnosticabilidade de grafos estrela considerando o modelo MM* é apresentada por Zheng, Latifi, Regentova, Luo e Wu [204].

Um grafo estrela n-dimensional, também referenciado por n-*star* ou S_n, é um grafo não direcionado que consiste de $n!$ nodos e $(n-1)n!/2$ arestas [2]. A cada nodo é assinalado um identificador único $i_1 i_2 \ldots i_m \ldots i_n$, que é uma permutação distinta de um conjunto de n símbolos $\{a_1, a_2, \ldots, a_n\}$. Sem perda de generalidade, seja o conjunto de n símbolos $\{a_1, a_2, \ldots, a_n\}$ o conjunto de inteiros $\{1, 2, \ldots, n\}$. Um nodo está conectado por uma aresta a outro nodo se e somente se o identificador de um dos nodos puder ser obtido através do identificador do outro nodo através da troca do primeiro símbolo de um pelo i-ésimo símbolo do outro, para $2 \leq i \leq n$. Em S_n cada nodo é conectado a $n-1$ nodos, isto é, cada nodo possui grau $n-1$. Além disso, cada S_n pode ser decomposto em n grafos estrela, cada um $(n-1)$-dimensional.

Como exemplo, em um 4-*star* contendo 4! nodos, dois nodos x com identificador 1234 e y com identificador 4231 são vizinhos e conectados através de uma aresta. Um grafo 4-*star* (S_4) é mostrado como exemplo na Figura A.9.

Zheng, Latifi, Regentova, Luo e Wu [204] usam as três condições suficientes apresentadas na caracterização de Sengupta e Dahbura [169] e mostram que um sistema com N nodos é t-diagnosticável se: (1) $N \geq 2t+1$; (2) cada nodo possui grau maior ou igual a t; (3) para cada $X \subset V$ tal que $|X| = N - 2t + p$ e $0 \leq p \leq t-1$ então $|T(X)| > p$.

Os autores também provam que um grafo estrela n-dimensional é $(n-1)$-diagnosticável para $n \geq 4$. Para provar, os autores mostram que um S_n satisfaz as três condições suficientes de diagnosticabilidade para $n \geq 4$, como segue. A primeira condição: como o número de nodos N em S_n é $n!$, então $n! \geq 2(n-1)+1$ é verdadeiro quando $n \geq 3$. A segunda condição segue do fato de que cada nodo de S_n possui grau $n-1$. Os autores

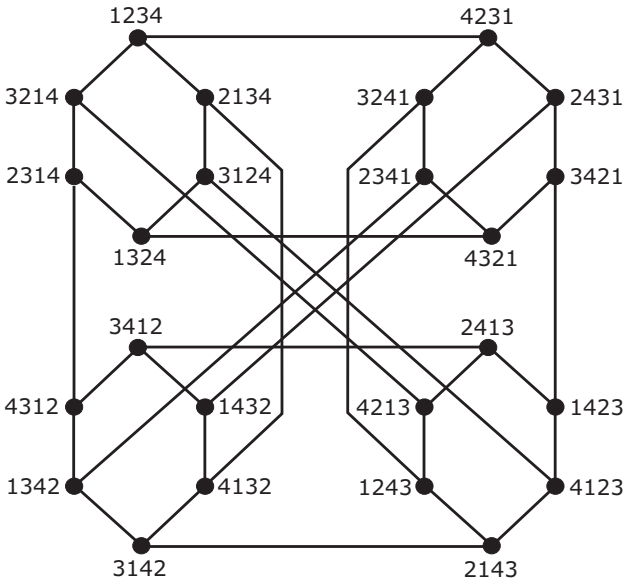


Figura A.9: Um exemplo de um grafo estrela 4-dimensional, S_4.

mostram a terceira condição em dois passos: primeiramente eles provam, por contradição, que para $p = n - 2$, para um arbitrário $X \subset V$ tal que $|X| = n! - 2(n-2) + p$ onde $0 \leq p \leq n-2$, então $|T(X)| > p$ é verdadeiro; então eles provam, também por contradição, que para $p = 0, 1, \ldots, n-3$ então $|T(X)| > p$ é verdadeiro.

Finalmente, ambos os algoritmo polinomiais apresentados em [169] e [198] podem ser aplicados em grafos estrelas n-dimensionais para encontrar o conjunto de nodos falhos do sistema, se o número de nodos falhos não for maior que $n - 1$.

A.6 Diagnóstico Baseado em Comparações para *Matching Composition Networks*

A diagnosticabilidade de *matching composition networks* é apresentada por [124] e também é baseada no modelo de diagnóstico baseado em comparações de Maeng e Malek. Uma *matching composition network* (MCN) é uma topologia de rede que consiste de dois componentes que são conectados por uma correspondência perfeita. Uma MCN inclui muitas topologias como casos especiais, como o hipercubo, o cubo cruzado, o *twisted cube*, e o *Möbius cube* [44, 79]. As MCNs podem ser construídas recursivamente. Elas são cons-

truídas a partir de dois grafos com o mesmo número de nodos, através da adição de uma correspondência perfeita entre os nodos dos dois grafos.

Uma MCN é um grafo $G = (V, E)$ definido como segue. Sejam $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ dois grafos com o mesmo número de nodos e todo nodo $v \in V_i$ de G_i possui $d_{G_i}(v) \geq t$, onde $i = 1, 2$ e $d_{G_i}(v)$ representa o grau do vértice v no grafo G_i. Seja L uma correspondência arbitrária, dita perfeita (*perfect matching*), entre os nodos V_1 de G_1 aos nodos V_2 de G_2, também denotada por $L(V_1, V_2)$. Em outras palavras, L é o conjunto das arestas conectando os nodos de G_1 aos nodos de G_2 em uma forma um-para-um. O grafo composto resultante é uma MCN; os grafos G_1 e G_2 são chamados de *componentes* da MCN.

Seja MCN_i uma MCN i-dimensional. MCN_1 é um grafo completo de dois vértices. Para $n \geq 2$, cada MCN_n consiste de duas MCN_{n-1}, denotadas por MCN^a_{n-1} e MCN^b_{n-1} com uma correspondência perfeita e arbitrária L. L é o conjunto de arestas que conecta MCN^a_{n-1} e MCN^b_{n-1}. O número de vértices na MCN_n é 2^n e cada um possui n vértices vizinhos.

Uma MCN é representada por $G(G_1, G_2; L)$ que possui o conjunto de nodos

$$V(G(G_1, G_2; L)) = V(G_1) \cup V(G_2)$$

e o conjunto de arestas

$$E(G(G_1, G_2; L)) = E(G_1) \cup V(G_2) \cup L.$$

Um exemplo de uma MCN_3, $G(G_1, G_2; L)$ é mostrado na Figura A.10.

Lai, Tan, Tsai e Hsu avaliam a diagnosticabilidade de *matching composition networks* sobre o modelo MM* [124]. No modelo apresentado, $M = (V, C)$ também é o multigrafo de comparações, e o grafo G representa a MCN. A notação $(u, v)_w$ também representa uma comparação, isto é, o nodo w compara as saídas de tarefas executadas pelos nodos u e v. Seja $U \in V$ e $\bar{U} = V - U$, $T(G, U)$ o conjunto $\{v \mid (u, v)_w \in C \text{ e } w, u \in U \text{ e } v \in \bar{U}\}$. Os autores mostram que uma MCN G com N nodos é t-diagnosticável se:

1. $N \geq 2t + 1$;

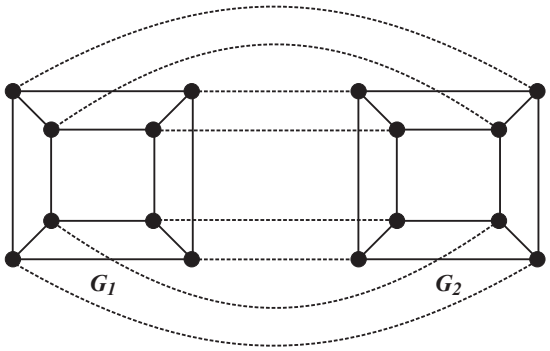


Figura A.10: Um exemplo de uma MCN_3, $G(G_1, G_2; L)$.

2. $d_G(v) \geq t$ para todo nodo v em G;

3. para todo par distinto de subconjuntos $S_1, S_2 \in V(G)$ tal que $|S_1| = |S_2| = t$, uma das seguintes condições é satisfeita:

   (a) $|T(G, U)| > p$, onde $U = V - (S_1 \cup S_2)$, e $|S_1 \cap S_2| = p$, ou

   (b) $\exists i, j \in S_1 - S_2$ e $\exists k \in V - S_1 - S_2$ tal que $(i, j)_k \in C$, ou

   (c) $\exists i, j \in S_2 - S_1$ e $\exists k \in V - S_1 - S_2$ tal que $(i, j)_k \in C$.

Os autores também provam que uma MCN $G(G_1, G_2; L)$ é $(t + 1)$-diagnosticável se $t \geq 2$, G_1 e G_2 são dois grafos com o mesmo número de nodos N, $N \geq t + 2$, e todo nodo v em G_i possui $d_{G_i}(v) \geq t$, onde $i = 1, 2$. Eles também provam que a diagnosticabilidade de hipercubos, de cubos cruzados, de *twisted cubes* e de *Möbius cubes* de n dimensões é n, para $n \geq 4$.

Araki e Shibata introduzem o (t, k)-diagnóstico em [12]. O modelo de diagnóstico (t, k) é uma generalização do modelo PMC [158] e do modelo BGM [15]. O (t, k)-diagnostico garante que ao menos k unidades falhas em um sistema são identificadas e reparadas em cada iteração desde que o número de unidades falhas não exceda t, onde $k \leq t$. Desta forma o (t, k)-diagnostico permite o diagnóstico correto, mas incompleto. O (t, k)-diagnostico é uma generalização que também inclui ambos os sistemas em um passo e sequencialmente diagnosticáveis: no diagnóstico em um passo $t = k$ e no diagnóstico sequencial $k = 1$.

Chang, Chen e Chang em [31] aplicaram o (t, k)-diagnóstico para *matching compo-*

sition networks sobre o modelo MM*. Eles provam que uma MCN de n dimensões é $(\Omega(\frac{2^n logn}{n}), n)$-diagnosticável, para $n > 5$. Eles estendem os seus resultados e provam que hipercubos, cubos cruzados, *twisted cubes* e *Möbius cubes* de n dimensões são todos $(\Omega(\frac{2^n logn}{n}), n)$-diagnosticáveis, para $n > 5$. Em [31] os autores também apresentam um algoritmo polinomial $O(|E|)$ para o (t, k)-diagnóstico sobre o modelo MM*.

Recentemente em [127] Lee e Hsieh avaliaram a diagnosticabilidade de *two-matching composition networks* sobre o modelo MM*. Uma *two-matching composition network* é definida como segue. Sejam G_1 e G_2 dois grafos com o mesmo número de vértices. Considere novamente L, uma correspondência perfeita entre os nodos de G_1 e G_2 – ou seja, L é o conjunto das arestas conectando os nodos de G_1 aos nodos de G_2 em uma forma um-para-um. Seja PM_2 um conjunto composto com duas correspondências perfeitas diferentes, entre os vértices de G_1 aos vértices de G_2. O grafo resultante construído a partir de G_1 e G_2, conectando cada vértice de G_1 a cada vértice de G_2 através de PM_2 é chamado de um *two-matching composition network*, ou ainda 2-MCN, e é denotado por $G(G_1, G_2; PM_2)$.

A Figura A.11 mostra um exemplo de uma 2-MCN. A figura destaca dentro dos círculos os grafos G_1 e G_2. Ambos os grafos possuem 4 nodos. O conjunto de arestas que conectam os vértices de G_1 aos vértices de G_2 representa a PM_2. A figura ainda diferencia cada uma das duas correspondências da PM_2, mostrando uma delas em linhas pontilhadas.

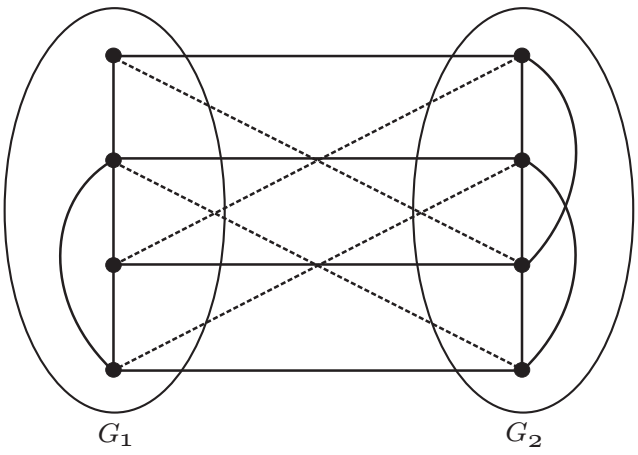


Figura A.11: Exemplo de uma 2-MCN criada a partir de dois grafos G_1 e G_2.

Lee e Hsieh provam que a diagnosticabilidade de um grafo $G(G_1, G_2; PM_2)$ é $t + 2$ desde que [127]: (i) G_1 e G_2 são grafos com N nodos, ambos com o mesmo número de nodos; (ii) $N \geq t + 3$; (iii) $t \geq 2$; e (iv) Se $d_{G_i}(v) \geq t$ então $\kappa(G_i) \geq t$ e $|N(v)| \geq 3$ para cada nodo de G_i, $i = 1, 2$; onde $\kappa(G) = min\{|V'|$ tal que $V' \subseteq V$ e $G - V'$ não é conectado$\}$, ou seja, $\kappa(G)$ é o tamanho do menor conjunto de vértices tal que quando removidos de G, o grafo resultante não é conexo – $\kappa(G)$ também pode ser chamado de *conectividade* de G.

Os autores ainda apresentam, também em [127], a diagnosticabilidade de dois casos especiais de *two-matching composition networks*: a diagnosticabilidade das topologias *augmented cubes* e *folded hypercubes*.

Um *n-dimensional augmented cube* – denotado por AQ_n – é definido de forma recursiva como segue [39]. Um AQ_1 é um grafo completo com dois nodos com rótulo respectivamente 0 e 1. Para $n \geq 2$, um AQ_n é obtido através da adição de $2 * 2^{n-1}$ arestas a duas cópias de AQ_{n-1} – que são denotadas respectivamente por AQ^0_{n-1} e AQ^1_{n-1} – da seguinte forma: Considere $V(AQ^0_{n-1}) = \{0a_{n-1}a_{n-2}\ldots a_1 | a_i \in \{0, 1\}\}$, e $V(AQ^1_{n-1}) = \{1b_{n-1}b_{n-2}\ldots b_1 | b_i \in \{0, 1\}\}$. Um nodo $u = 0a_{n-1}a_{n-2}\ldots a_1 \in V(AQ^0_{n-1})$ é conectado a um nodo $v = 1b_{n-1}b_{n-2}\ldots b_1 \in V(AQ^1_{n-1})$ se e somente se:

(i) $a_i = b_i$ for $1 \leq i \leq n - 1$, ou

(ii) $a_i = \overline{b_i}$ for $1 \leq i \leq n - 1$.

Como exemplo, a Figura A.12 mostra três cubos aumentados – um AQ_1, um AQ_2 e um AQ_3.

Lee e Hsieh então provam que a diagnosticabilidade dos cubos aumentados n-dimensionais AQ_n, com base no modelo MM*, é $2n - 1$ para $n \geq 5$ [127].

Por sua vez, um *n-dimensional folded hypercube* – denotado por FQ_n – é definido da seguinte forma [63]. Um cubo FQ_n é equivalente ao hipercubo de mesma dimensão (H_n), com a adição de mais $N/2$ arestas (ou enlaces) extras ao conjunto de $n * 2^{n-1}$ arestas já existente no H_n, resultando em um total de $(n + 1)2^{n-1}$ arestas. Além disso, no FQ_n

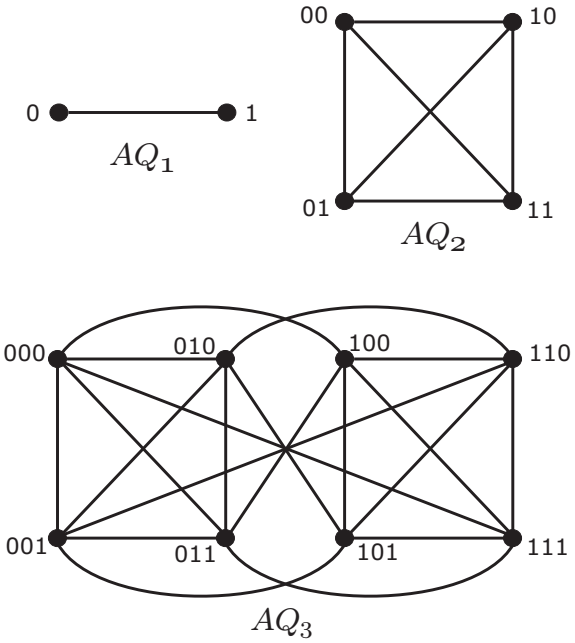


Figura A.12: Exemplos de um AQ_1, um AQ_2 e um AQ_3.

um nodo u é conectado ao nodo v, isto é, existe a aresta (u, v), somente se a distância de Hamming entre u e v – $H(u, v)$ – for 1 ou n. Em outras palavras, considerando o identificador binário dos nodos u e v, u é conectado a v somente se apenas 1 *bit* for diferente ou se *todos* os *bits* forem diferentes, isto é, neste último caso se u for o complemento de v.

A Figura A.13 mostra um exemplo de um FQ_3. No exemplo as linhas pontilhadas representam as arestas extras adicionadas ao n-hipercubo correspondente.

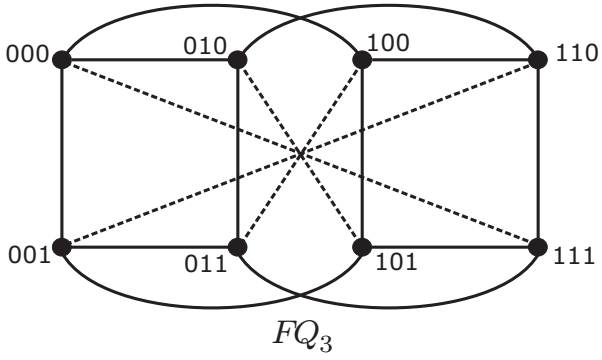


Figura A.13: Exemplos de um FQ_3. As linhas pontilhadas representam as arestas extras adicionadas.

Por fim, os autores provam então que a diagnosticabilidade dos *folded hypercubes* FQ_n, com base no modelo MM*, é $n + 1$ para $n \geq 4$ [127].

A.7 Diagnóstico Baseado em Comparações para Redes t-Conectadas e Redes Produto

A diagnosticabilidade de redes t-conectadas (t-*conected networks*) e redes produto (*product networks*) sobre o diagnóstico baseado em comparações foi apresentada por Chang, Lai, Tan e Hsu em [29] também sobre o modelo MM*. Um grafo G é t-conectado se $\kappa(G) \geq t$ onde $\kappa(G) = min\{|V'|$ tal que $V' \subseteq V$ e $G - V'$ não é conectado$\}$.

Uma rede produto é gerada pela aplicação da operação de produto cartesiano de grafos a redes de fator. Uma rede de produto cartesiano (*cartesian product network*) $G = G_1 \times G_2$ [9] de dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ é o grafo $G = (V, E)$. Os grafos G_1 e G_2 são chamados os *fatores* ou *redes componentes* do grafo G. O conjunto de nodos V e o conjunto de arestas E de G são dados por:

1. $V = \{\langle x, y\rangle \mid x \in V_1$ e $y \in V_2\}$, e
2. para $u = \langle x_u, y_u\rangle$ e $v = \langle x_v, y_v\rangle$ in $V, (u, v) \in E$ se e somente se $(x_u, x_v) \in E_1$ e $y_u = y_v$, ou $(y_u, y_v) \in E_2$ e $x_u = x_v$.

Como exemplo, a Figura A.14 mostra dois grafos de redes G_1 e G_2 e o grafo da rede de produto cartesiano correspondente $G_1 \times G_2$.

Chang, Lai, Tan e Hsu avaliam a diagnosticabilidade destas topologias também assumindo as condições apresentadas por Sengupta e Dahbura [169]. Eles mostram que uma rede t-regular e t-conectada com N nodos e $t > 2$ é t-diagnosticável se $N \geq 2t + 3$. Além disso, a rede produto de G_1 e G_2 se mostra ser $(t_1 + t_2)$-diagnosticável, onde G_i é t_i-conectado para $i = 1, 2$.

A.8 *Strong Diagnosability* para Diagnóstico Baseado em Comparações

A diagnosticabilidade forte (*strong diagnosability*) de sistemas sobre o modelo PMC foi primeiramente apresentada por Lai, Tan, Chang e Hsu em [125]. Um sistema é fortemente

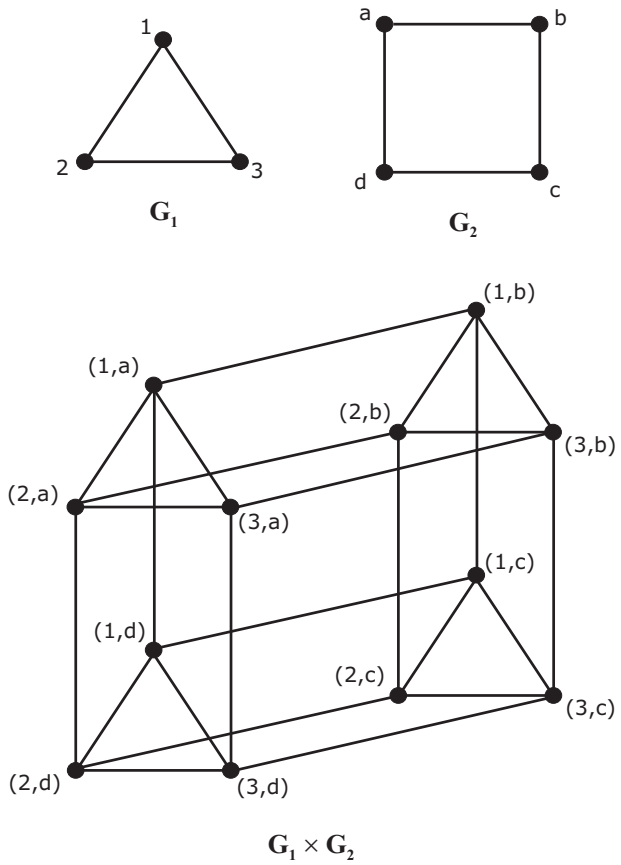


Figura A.14: Duas redes G_1 e G_2 e a rede produto correspondente $G_1 \times G_2$.

t-diagnosticável se ele for $(t+1)$-diagnosticável e não existe um nodo tal que todos os seus vizinhos sejam falhos. Em outras palavras: a diagnosticabilidade forte mostra a habilidade de um sistema t-diagnosticável em detectar $t+1$ nodo falho, assumindo que todos os vizinhos de qualquer nodo não podem falhar simultaneamente. O valor t tal que o sistema é fortemente t-diagnosticável também é representado por $t_s(G)$, isto é, $t_s(G) = t$ se o sistema é fortemente t-diagnosticável.

Sheu, Huang e Chen [171] foram os primeiros a investigar a diagnosticabilidade forte de sistemas sobre o modelo MM*. Seja uma rede t-regular com grau $d(u) = t$ para todo nodo u. Os autores mostram que uma rede t-regular e t-conectada na qual $N \geq 2t+6$ e $t \geq 4$ é fortemente t-diagnosticável se o sistema é livre de triângulos e a interseção do conjunto de vizinhos de qualquer par de nodos no sistema possui no máximo $t-2$ nodos.

Hsieh e Chen [104] investigam a diagnosticabilidade forte para uma classe de redes produto sobre o modelo MM*. Como definido na Seção A.7, uma rede produto é gerada

através da aplicação da operação de produto cartesiano a redes de fator. As redes produto incluem topologias como os hipercubos, *mesh-connected k-ary n-cubes*, *torus-connected k-ary n-cubes*, e redes *hyper-Petersen*. Redes produto regulares podem ser classificadas em duas subclasses: redes produto homogêneas e redes produto heterogêneas. Redes produto homogêneas são t-diagnosticáveis e t-regulares, enquanto as redes produto heterogêneas são compostas de duas diferentes redes de fator, onde uma é t-diagnosticável e a outra é t-conectada.

Para $t_i > 3$, a diagnosticabilidade forte de redes produto homogêneas $G_1 \times G_2 \times ... \times G_k = t_1 + t_2 + ... + t_k$, onde $G_i = (V_i, E_i)$ é uma rede t_i-diagnosticável e t_i-regular com N_i nodos, e $i = 1, 2, ..., k$. Considere que $G_i = (V_i, E_i)$ é uma rede t_i-diagnosticável e t_i-regular com N_i nodos para $i = 1, ..., m$ e seja $G_j = (V_j, E_j)$ uma rede t_j-conectada e t_j-regular com $N_j \geq 2t_j+1$ nodos para $j = m+1, ..., k$. Para $t_i > 3$, se $G = G_1 \times G_2 \times ... \times G_k$, então a diagnosticabilidade forte de G é $t_1 + t_2 + ... + t_k$. Para a diagnosticabilidade forte de redes produto não regulares, considere que $G_1 = (V_1, E_1)$ é t_1-diagnosticável, L_{k_i} é um *array* linear k_i-nodo, e $k_i \geq 2$ para $1 \leq i \leq l$. Os autores provam que, para $t_i > 3$, a rede produto não regular $G = G1 \times L_{k_1} \times L_{k_2} \times ... \times L_{k_l}$ é fortemente $(t_1 + l)$-diagnosticável.

A t-diagnosticabilidade forte de quatro diferentes topologias de redes produto, onde todas são t-regulares e t-conectadas é mostrada em [104]: o hipercubo n-dimensional, o *mesh-connected k-ary n-cube*, o *torus-connected k-ary n-cube*, e finalmente a rede *hyper-Petersen* n-dimensional. Para todas estas redes, $N \geq 2t + 1$ nodos, onde $t > 2$; cada nodo v de G possui grau maior ou igual a t. O primeiro resultado apresentado para a diagnosticabilidade forte foi para o hipercubo n-dimensional, que é n para $n \geq 5$. As outras três topologias e seus resultados para a diagnosticabilidade forte são apresentados abaixo.

Um *mesh-connected k-ary n-cube* [18], denotado por M_k^n, é recursivamente definido como segue: seja L_k um array linear de tamanho k, (1) $M_k^1 = L_k$, para $k \geq 2$, e (2) $M_k^n = M_k^{n-1} \times L_k$ para $n \geq 2$. Um M_k^n possui k^n nodos. Como exemplo, a Figura A.15 mostra um M_4^2. Os autores provam que a diagnosticabilidade forte de $M_k^n = n$ para $n \geq 5$.

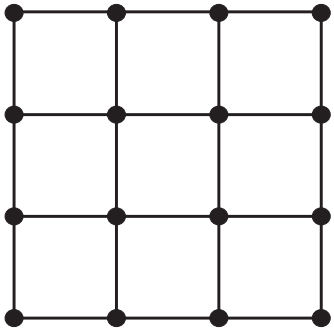


Figura A.15: Exemplo de um M_4^2.

Um *torus-connected k-ary n-cube* [18], denotado por T_k^n, é recursivamente definido como segue: seja R_k um anel (um ciclo) de tamanho k, onde $k \geq 3$. Então, (1) $T_k^1 = R_k$, e (2) $T_k^n = T_k^{n-n} \times R_k$ para $n \geq 2$. Um T_k^n também possui k^n nodos. A Figura A.16 mostra um exemplo de T_4^2. A diagnosticabilidade forte de um *torus-connected k-ary n-cube* é $2n$ para $k \geq 3$ e $n \geq 4$.

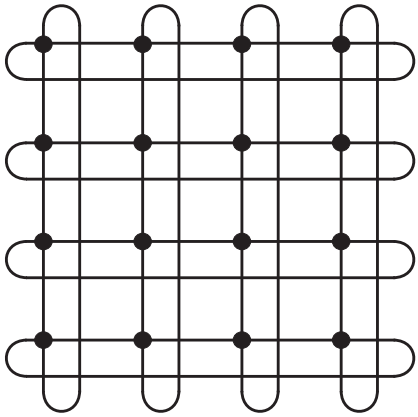


Figura A.16: Exemplo de um T_4^2.

Uma rede *hyper-Petersen* n-dimensional [48], denotada por HP_n para $n \geq 3$, é definida como $HP_n = P \times Q_{n-3}$, onde P é um grafo *Petersen*. Um HP_n é n-conectado e n-regular e possui $10 * 2^{n-3}$ nodos. A Figura A.17 mostra um exemplo de HP_4. A diagnosticabilidade forte de $HP_n = n$ para $n \geq 5$.

Posteriormente, Hsieh e Chen apresentaram em [105] a diagnosticabilidade forte para uma série de topologias, que são abrangidas pela classe das *matching composition networks* (MCN), sobre o modelo MM*. Eles avaliaram a diagnosticabilidade forte de cubos cru-

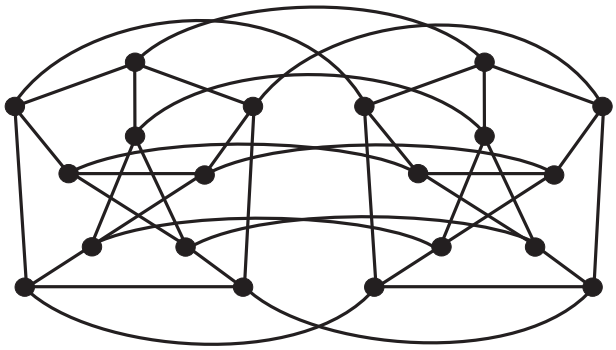


Figura A.17: Exemplo de um HP_4.

zados n-dimensional, *Möbius cubes*, *twisted cubes* e *locally twisted cubes*. Um cubo cruzado n-dimensional CQ_n é fortemente n-diagnosticável para $n \geq 5$. Um *Möbius cube* n-dimensional MQ_n é fortemente n-diagnosticável para $n \geq 5$. Um *twisted cube* n-dimensional TQ_n é fortemente n-diagnosticável para um inteiro impar $n \geq 5$. Finalmente, um *locally twisted cube* n-dimensional LTQ_n é fortemente n-diagnosticável para $n \geq 4$.

Mais recentemente, em [102] Hong e Hsieh também consideram o modelo MM* para determinar a diagnosticabilidade forte sobre os cubos aumentados n-dimensionais (n-*dimensional augmented cubes*), ou AQ_n. Uma introdução, incluindo a definição de construção dos cubos aumentados já foi apresentada na Seção A.6 deste anexo. Hong e Hsieh provam então que nos AQ_n, a diagnosticabilidade forte é $(2n-1)$ para $n \geq 5$.

Já em [107] os autores apresentam, também para o modelo MM*, as condições suficientes para determinar se um sistema com até t nodos falhos possui diagnosticabilidade forte. Algumas definições usadas para analisar as condições de diagnosticabilidade são descritas a seguir.

Considerando um sistema com N nodos representado por um grafo $G = (V, E)$, um subconjunto $I \in V$ é um *conjunto independente* de G se nenhum par de vértices de I são adjacentes em G. O *número de independência* (*independence number*) de G, denotado por $\alpha(G)$, é o tamanho do maior conjunto independente de vértices de G. Além disso, δ é o grau da unidade de menor grau do sistema e $\kappa(G) = min\{|V'|$ tal que $V' \subseteq V$ e $G - V'$ não é conectado$\}$, ou seja, $\kappa(G)$ é o tamanho do menor conjunto de vértices tal que quando removidos de G, o grafo resultante não é conexo.

Os autores então provam que um sistema é fortemente t-diagnosticável sobre o modelo MM* se as três seguintes condições forem satisfeitas:

(i) $N - 2t - 3 \geq \alpha(G)$;

(ii) $\kappa(G) = \delta = t$;

(iii) para qualquer conjunto $X \subset V$ onde $|X| = t$, se o grafo resultante da remoção dos vértices X de G não é conectado, então deve existir um nodo $u \in V$ tal que $N(u) \subseteq X$.

Também em [107] os autores consideram novamente o modelo MM* para determinar o valor t para a diagnosticabilidade forte sobre os *folded hypercubes* FQ_n. A definição dos *folded hypercubes* já foi apresentada na Seção A.6. Os autores provam que a diagnosticabilidade forte dos FQ_n sobre o modelo MM* é $n + 1$ para $n \geq 5$.

A.9 *Conditional Diagnosability* para Diagnóstico Baseado em Comparações

A diagnosticabilidade condicional (*conditional diagnosability*) de sistemas também foi primeiramente apresentada por Lai, Tan, Chang e Hsu em [125]. Um sistema $G = (V, E)$ é *condicionalmente* t-diagnosticável (*conditionally t-diagnosable*) se F_1 e F_2 são distinguíveis, para cada par *conjuntos condicionais de unidades falhas* F_1 e F_2, tal que $F_1, F_2 \subset V$ e $F_1 \neq F_2$, onde $F_1 \leq t$ e $F_2 \leq t$. Por sua vez, um conjunto de unidades falhas $F \subset V$ é um *conjunto condicional de unidades falhas* se, para todo $v \in V$, $N(v) \nsubseteq F$. Em outras palavras, um conjunto de unidades falhas F é condicional se não existe no sistema um nodo v tal que todos os seus vizinhos sejam falhos. O trabalho utiliza as notações $t(G)$ e $t_c(G)$ para representar, respectivamente, o valor t tal que o sistema é t-diagnosticável e condicionalmente t-diagnosticável. Argumenta-se ainda que a diagnosticabilidade condicional tem estreita relação com a diagnosticabilidade forte, mas não se limita ao diagnóstico de $(t + 1)$ unidades falhas, ou seja, tem o objetivo de determinar

qual o maior valor $t_c(G)$ tal que o sistema seja t-diagnosticável. Os autores provam que considerando um sistema G, claramente $t_c(G) \geq t(G)$.

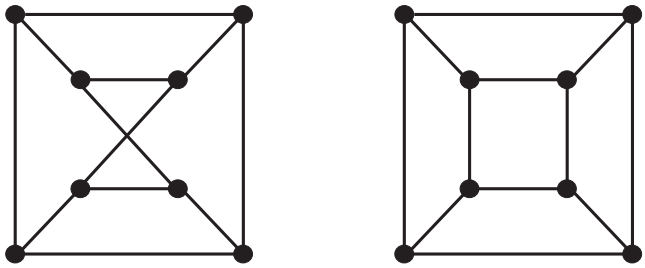
Em [109], Hsu e Tan avaliam a diagnosticabilidade condicional de redes BC (ou grafos BC – *Bijective Connection graphs*) sobre o modelo de diagnóstico baseado em comparações apresentado por Maeng e Malek.

Uma rede BC n-dimensional (*n-dimensional BC Network*) [206, 207], é denotada por X_n; já o conjunto de todas as redes BC n-dimensionais é chamado de família de redes BC n-dimensionais, e é denotado por $\mathbb{L}_n$. Ambos X_n e $\mathbb{L}_n$ são definidos de forma recursiva, como segue. A rede BC 1-dimensional X_1 é um grafo completo com 2 vértices. A família das redes BC 1-dimensionais $\mathbb{L}_1 = \{X_1\}$. Agora considere que o grafo $G_x = (V_x, E_x)$ é um subgrafo de $G = (V, E)$ *induzido por* V_x – denotado por $G[V_x]$ – se $E_x = \{(u, v) \in E \mid u, v \in V_x\}$. Um grafo $G = (V, E)$ pertence à $\mathbb{L}_n$ se e somente se existe dois conjuntos de vértices $V_0, V_1 \subset V$ tal que as seguintes condições são satisfeitas:

(i) $V = V_0 \cup V_1$, $V_0 \neq \emptyset$, $V_1 \neq \emptyset$, $V_0 \cap V_1 = \emptyset$, e $G[V_0], G[V_1] \in \mathbb{L}_{n-1}$;

(ii) O conjunto de arestas de G que conectam os vértices V_0 aos vértices de V_1 é uma correspondência perfeita L.

Pela definição acima, se para qualquer rede BC $X_n \in \mathbb{L}_n$, então existe V_0, V_1 e L que satisfazem as duas condições acima. Além disso, os grafos induzidos $G[V_0]$ e $G[V_1]$ são grafos X_{n-1}, ou seja, redes BC $(n-1)$-dimensionais. As Figuras A.18(a) e A.18(b) mostram dois exemplos de redes BC 3-dimensionais X_3.



(a) (b)

Figura A.18: Exemplos de duas redes BC X_3.

Os autores de [109] então provam que, com base no diagnóstico baseado em comparações, a diagnosticabilidade condicional de redes BC n-dimensionais é $3(n-2)+1$ para $n \geq 5$.

Já os autores de [108] avaliam a diagnosticabilidade condicional de hipercubos n-dimensionais sobre o modelo MM de diagnóstico baseado em comparações. Os autores provam que a diagnosticabilidade condicional dos n-hipercubos é $3(n-2)+1$ para $n \geq 5$. Eles ainda enfatizam que, no modelo MM, a diagnosticabilidade condicional dos n-hipercubos é cerca de três vezes maior do que a t-diagnosticabilidade para os mesmos hipercubos. Em [205] avalia-se a diagnosticabilidade condicional de cubos cruzados CQ_n, também sobre o modelo MM. Os autores mostram que diagnosticabilidade condicional dos CQ_n é $3n-5$ quando $n \geq 7$. Os autores também apontam que os cubos cruzados, de forma similar ao que ocorre nos hipercubos, também possuem diagnosticabilidade condicional três vezes maior do que a t-diagnosticabilidade.

Em [106] a diagnosticabilidade condicional agora com base no modelo MM*, sobre os *k-ary n-cubes*, um classe de topologias especiais que incluem, entre outros, os já apresentados n-hipercubos, *mesh-connected k-ary n-cubes*, e *torus-connected k-ary n-cube*. Os autores provam que a diagnosticabilidade condicional dos *k-ary n-cubes* sobre o modelo MM* é $6n-5$ para $k \geq 4$ e $n \geq 4$.

Em [102] Hong e Hsieh também consideram o modelo MM* para determinar a diagnosticabilidade condicional sobre os cubos aumentados n-dimensionais (AQ_n). Uma definição dos cubos aumentados AQ_n já foi apresentada na Seção A.6. Hong e Hsieh provam então que nos AQ_n, a diagnosticabilidade condicional considerando o modelo MM* é $6n-17$ para $n \geq 6$.

Recentemente em [107] os autores também apresentam as condições suficientes para determinar se um sistema com até t nodos falhos é condicionalmente t-diagnosticável sobre o modelo MM*. Os autores usam as seguintes definições para apresentar a diagnosticabilidade do sistema: considerando um sistema com N nodos representado por um grafo $G = (V, E)$, um subconjunto $I \in V$ é um *conjunto independente* de G se nenhum par de

vértices de I são adjacentes em G. O *número de independência* (*independence number*) de G, denotado por $\alpha(G)$, é o tamanho do maior conjunto independente de vértices de G.

Os autores então provam que um sistema é condicionalmente t-diagnosticável sobre o modelo MM* se as duas seguintes condições forem satisfeitas:

(i) $N - 2t - 1 \geq \alpha(G)$, e

(ii) para qualquer conjunto $X \subset V$ onde $|X| \leq t - 1$, o grafo resultante da remoção dos vértices X de G é conectado.

Além disso, também em [107] os autores determinam a diagnosticabilidade forte sobre os *folded hypercubes* FQ_n, novamente com base no modelo MM*. A definição dos *folded hypercubes* também já foi apresentada na Seção A.6. Os autores então provam que a diagnosticabilidade condicional dos FQ_n é $3n-2$ para $n \geq 5$; além disso, a diagnosticabilidade condicional para os FQ_3 e FQ_4 são respectivamente 3 e 7.

A.10 Diagnóstico Baseado em Comparações com *Broadcast*

O modelo de diagnóstico baseado em comparações com *broadcast* foi apresentado por Blough e Brown em [22]. Este modelo aplica o diagnostico distribuído baseado no modelo de comparações MM* [140] para sistemas que possuem um serviço de *broadcast* confiável fraco (*weak reliable broadcast*). Neste modelo, um procedimento distribuído de diagnóstico é utilizado, que por sua vez também é baseado na comparação de saídas de tarefas redundantes.

O sistema é também modelado como um grafo $G = (V, E)$. Tarefas são enviadas a pares de diferentes nodos. Os dois nodos executam a tarefa e as saída da tarefas são enviadas para todos os nodos usando um serviço de *broadcast* confiável. Depois que as saídas das tarefas são recebidas, elas são comparadas com o objetivo de detectar falhas. As comparações são realizadas por todos os nodos do sistema. A Figura A.19 exemplifica este procedimento. Nesta figura o nodo 1 envia uma mesma tarefa ao nodo 2 e ao nodo

3, que por sua vez executam a tarefa e fazem o *broadcast* das saídas para todos os nodos do sistema.

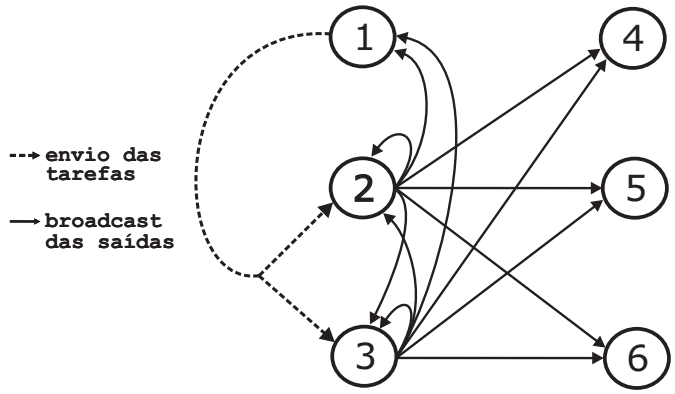


Figura A.19: Uma tarefa é enviada a partir do nodo 1 para os nodos 2 e 3. Ocorre *broadcast* das saídas das tarefas para todos os nodos do sistema.

Todo nodo sem-falha do sistema compara as duas saídas produzidas, incluindo os próprios nodos que produziram as saídas. A síndrome é a coleção completa dos resultados de todas as comparações. Assim que cada nodo executar todas as comparações, ele completa o diagnóstico do sistema assumindo a si próprio como sem-falha.

As principais asserções do modelo de comparações baseado em *broadcast* são:

1. Quando dois nodos sem-falha executam a mesma tarefa, eles produzem a mesma saída, e a comparação destas duas saídas realizadas por nodos sem-falha do sistema resulta em igualdade.
2. Um nodo falho sempre produz uma saída para uma tarefa que resulta em diferença quando comparada a uma saída de uma tarefa produzida por qualquer outro nodo falho ou sem-falha.
3. O *broadcast* de qualquer mensagem de um processador sem-falha é corretamente recebido por todos os processadores sem-falha em um tempo limitado.
4. O tempo para que qualquer tarefa gere uma saída é limitado.
5. Cada processador possui um identificador único.

6. Processadores sem-falha podem corretamente identificar quem realizou o *broadcast* de uma mensagem.

7. Saídas enviadas por processadores falhos são corretamente recebidas por processadores sem-falha; além disso a comparação realizada por um processador sem-falha de um processador falho e qualquer outro processador sempre resulta em diferença.

As asserções (1) e (2) são herdadas dos modelos MM e MM*. As outras asserções são construídas com o objetivo de garantir as duas primeiras asserções. A asserção (3) é a asserção básica do *broadcast* confiável fraco. Um *broadcast* confiável fraco [93] requer que processadores sem-falha recebam todas as mensagens, mesmo aquelas enviadas por um processador falho, mas este tipo de *broadcast* não possui qualquer requerimento sobre a ordem das mensagens. A asserção (7) evita que saídas sejam modificadas durante a comunicação.

Blough e Brown apresentam um algoritmo polinomial para a análise da diagnosticabilidade do sistema sobre o modelo de comparações baseado em *broadcast*. Cinco definições são necessárias para caracterizar a diagnosticabilidade do sistema:

1. Um conjunto independente no grafo $G = (V, E)$ é um subconjunto $V' \subseteq V$ tal que, para todo $u, v \in V'$, $(u, v) \notin E$.

2. Para o grafo $G = (V, E)$ e um processador $u \in V$, $N(u) = \{v \in V \mid (u, v) \in E\}$, isto é, o conjunto de vizinhos do processador u. Também, $|N(u)| = d(u)$.

3. Para o grafo $G = (V, E)$ e um conjunto $Z \in V$, $N(Z) = \{v \in V - E \mid \exists u \in Z \text{ e } (u, v) \in E\}$, isto é, o conjunto de vizinhos de Z.

4. Para o grafo $G = (V, E)$, P_G é o conjunto de partições de V em quatro conjuntos disjuntos e par-relacionados (X, Y, Z_1, Z_2) tal que: (1) $X \neq \emptyset$; (2) $N(X) \subseteq Y$; (3) $Z_1 \cup Z_2 \neq$; e (4) Z_1 e Z_2 são conjuntos independentes.

5. Para o grafo $G = (V, E)$, κ é uma função de P_G para o conjunto de inteiros positivos tal que, para todo $p = (X, Y, Z_1, Z_2) \in P_G, \kappa(p) = |Y| + max(|Z_1|, |Z_2|)$.

Um sistema $G = (V, E)$ é t-diagnosticável se e somente se para todo $p \in P_G, \kappa(p) > t$.

A diagnosticabilidade de um sistema de N processadores para o grafo completo de comparações é $N - 1$. A diagnosticabilidade de um sistema que não possui o grafo completo de comparações disponível é $d_{min}(G)$ ou $d_{min}(G) - 1$, onde o grau $d(u)$ de um processador u em G é o número de arestas de G incidentes em u. O grau do nodo de menor grau no sistema G é $d_{min}(G) = min_{u \in V} d(u)$.

Em [22], Blough e Brown também apresentam um algoritmo polinomial para diagnosticar situações estáticas e dinâmicas de falhas usando o modelo de comparações baseado em *broadcast*. Em uma situação de falhas estáticas, nenhuma falha ocorre no sistema a partir do momento em que as comparações se iniciam e até que o diagnóstico chegue ao fim. Blough e Brown apresentam o algoritmo *Static-Complete* para o diagnóstico do sistema sobre a situação de falhas estáticas, a partir da síndrome completa.

A Figura A.20 mostra o algoritmo *Static-Complete*. O algoritmo executa em cada nodo x do sistema e recebe como entrada a síndrome do sistema e a diagnosticabilidade t. Cada nodo assume a si mesmo como sem-falha, se adicionando ao conjunto dos nodos sem-falha FF; isso é representado no passo 1. No passo 2, qualquer processador que possui o resultado de uma comparação que indique igualdade é adicionado ao conjunto FF. Se o número de processadores restantes (fora do conjunto FF) for no máximo t, o passo 3 termina o algoritmo. Caso contrário, o passo 4 identifica processadores falhos em FF e os adiciona ao conjunto F. O passo 5 determina se existe algum processador falho que ainda se encontra não identificado e então os adiciona ao conjunto F usando a função *Find_Remaining* que é mostrada na Figura A.21. Finalmente, no passo 6 o algoritmo termina quando o conjunto FF é obtido.

Em algumas situações onde o número de nodos falhos é muito menor que t, é ainda possível para processadores sem-falha diagnosticar corretamente o sistema sem realizar

Algoritmo: Static-Complete
/* *Entrada*: A síndrome do sistema e a diagnosticabilidade t */
/* *Saída*: O conjunto FF (fault-free) e o conjunto F (faulty) */
1) $F \leftarrow \emptyset$; $FF \leftarrow \{x\}$;
2) **para** cada aresta (u, v) com resultado de comparação igual a 0
   $FF \leftarrow FF \cup \{u, v\}$;
3) **se** $|V| - |FF| \leq t$ **então**
   $F \leftarrow V - FF$; **terminar**;
4) **para** cada aresta (u, v) com resultado de comparação igual a 1
   se $u \in FF$ **então** $F \leftarrow F \cup \{v\}$;
   se $v \in FF$ **então** $F \leftarrow F \cup \{u\}$;
5) **se** $|F| < t$ **então**
   $F \leftarrow F \cup Find\_Remaining(G[V - FF - F], t - |F|)$;
6) $FF \leftarrow V - F$;

Figura A.20: Algoritmo *Static-Complete*.

Função: Find_Remaining
/* *Entrada*: Um grafo $\hat{G}(\hat{V}, \hat{E})$ e um inteiro $\hat{t}$ tal que $0 < \hat{t} < |\hat{V}|$ */
/* *Saída*: Um conjunto de unidades falhas $\hat{F}$ */
1) **para** cada $u \in \hat{V}$ com resultado de comparação igual a 0
2) **se** $|N(u)| = \hat{t}$ **então**
3) $\hat{F} \leftarrow \{u\}$;
4) **para** cada $v \in \hat{V} - N(u) - \{u\}$
5) **se** $N(v) = N(U)$ **então** $\hat{F} \leftarrow \hat{F} \cup \{v\}$;
6) **se** $|\hat{F}| = |\hat{V}| - \hat{t}$ **então** retornar $N(u)$;

Figura A.21: Função *Find_Remaining*.

todas as comparações – esta situação é referenciada como diagnóstico através de uma síndrome parcial. O algoritmo *Static-Partial* é apresentado para situações onde somente uma síndrome parcial está disponível. Nestes casos nenhum algoritmo consegue garantir o diagnóstico do estado de todos os processadores, isto é, garante-se que o diagnóstico é correto, mas ele pode ser incompleto.

Como em sistemas reais falhas podem ocorrer durante a execução do algoritmo de diagnóstico, Blough e Brown apresentaram o algoritmo *Dynamic* para diagnosticar sistemas sobre situações dinâmicas de falhas. Entretanto, eles assumem que, uma vez que um processador falha, ele continua falho até a próxima execução do algoritmo de diagnóstico. Além disso, este modelo permite que processadores sem-falha fiquem falhos, enquanto não é permitido que processadores falhos se tornem sem-falha durante a execução do diagnóstico. As saídas das tarefas recebem selos cronológicos antes de sofrerem *broadcast* e o relógio dos processadores sem-falha deve avançar em uma taxa aproximadamente correta

e com *drift* limitado.

A principal diferença do modelo de comparações baseado em *broadcast* e o modelo MM* é que o modelo baseado em *broadcast* é completamente distribuído, enquanto que o modelo MM* se baseia em um observador central que recebe o resultado das tarefas e realiza todas as comparações. No modelo de comparações com *broadcast* todos os processadores sem-falha produzem o mesmo conjunto de resultados de comparações, ou seja, todos os processadores sem-falha produzem a mesma síndrome.

O principal propósito do modelo de comparações baseado em *broadcast* é reduzir a latência e o tempo em que os nodos precisam permanecer em um determinado estado, e não reduzir o número de testes ou o número de comparações executadas. O sistema precisa possuir já disponível uma primitiva (por exemplo, implementada em hardware) equivalente ao *broadcast* confiável. O sistema foi implementado no *COmmon Spaceborne Multicomputer Operating System* (COSMOS). Os autores também apresentam resultados obtidos com um simulador para o sistema multi-computador JPL MAX executando o COSMOS.

A.11 Diagnóstico Probabilístico Baseado em Comparações

Os modelos probabilísticos baseados em comparações foram primeiramente apresentados por Dahbura, Sabnani e King [46]. Todos estes modelos assumem uma probabilidade de falhas, isto é, a probabilidade de uma unidade produzir a saída incorreta; a diagnosticabilidade é calculada com base nesta probabilidade. Portanto, estes modelos não impõem um limite superior sobre o limite de unidades falhas no sistema.

Existem duas abordagens probabilísticas básicas para resolver o problema do diagnóstico. Estas abordagens foram propostas inicialmente para o diagnóstico clássico em nível de sistema; o diagnóstico probabilístico baseado em comparações apareceu posteriormente. A primeira abordagem é restringir o diagnóstico a um conjunto de unidades falhas, com uma probabilidade suficientemente alta [87, 142]. A outra abordagem é realizar o diagnóstico

para todo o sistema, e então provar que o diagnóstico é correto com uma alta probabilidade [24, 24, 25, 46, 163]. Em muitos casos, estes modelos refletem o ambiente real de falhas de uma maneira mais precisa, mas eles são geralmente mais difíceis de analisar.

No modelo de diagnóstico probabilístico baseado em comparações proposto por Dahbura, Sabnani e King [46], o sistema é também representado por um grafo $G = (V, E)$. Tarefas também são enviadas para pares de unidades e as saídas das tarefas são comparadas para identificar as unidades falhas. A coleção de todas as saídas é também chamada de síndrome do sistema. As asserções básicas do sistema são:

— m é o número total de diferentes possíveis saídas incorretas que um processador falho pode produzir para uma tarefa;

— $W_i \mid 1 \leq i \leq m$ é uma das m possíveis saídas incorretas para uma tarefa;

— $P(W_i)$ é a probabilidade de que uma unidade falha produza a saída incorreta W_i para uma tarefa; e,

— p é a probabilidade de que uma unidade falha produza a saída correta para uma tarefa.

Os seguintes resultados são obtidos a partir da avaliação deste modelo [46]:

1. a probabilidade $P_{1,0}$, de que a comparação de duas saídas indique igualdade, é igual a p quando uma das unidades que produziu a saída é falha, e
2. a probabilidade $P_{2,0}$, de que a comparação de duas saídas indique igualdade, é igual a $p^2 + P(W_1)^2 + ... + P(W_m)^2$ quando ambas as unidades que produziram as saídas são falhas.

Os autores assumem que a distribuição de probabilidades para uma unidade produzir resultado incorreto é uniforme; então $\forall i, P(W_i) = (1 - p)/m$. Assim, a probabilidade de que a comparação das saídas de duas unidades falhas resulte em igualdade é $P_{2,0} = p^2 + ((1 - p)^2/m)$. Além disso, assume-se que m é extremamente grande, então $P_{2,0} \approx p^2$.

Outro modelo probabilístico e baseado em comparações foi proposto por Pelc em [156]. Neste modelo, também chamado de modelo (p, k)-probabilístico, a mesma tarefa com k saídas possíveis é enviada às unidades. Cada unidade possui a mesma probabilidade $p < 1/2$ de se tornar falha e a falha de unidades distintas ocorre de forma independente. Este modelo assume que:

— unidades sem-falha sempre retornam saídas incorretas; e,

— unidades falhas retornam saídas incorretas de forma independente, com uma probabilidade uniforme $1/k$ para cada uma, mas eventualmente as saídas de duas unidades falhas podem ser idênticas.

Assim como nos primeiros modelos baseados em comparações, as saídas das tarefas são comparadas e o resultado, igualdade (0) ou diferença (1), é então usado para identificar as unidades falhas no sistema. A probabilidade de uma igualdade ser o resultado da comparação das saídas produzidas por duas unidades, uma sem-falha e outra falha, ou então por duas unidades falhas, é $q = 1/k$. Esta é a diferença deste modelo para o modelo proposto por Dahbura, Sabnani e King, no qual a probabilidade de se obter uma resposta incorreta de um processador falho é muito menor do que a da resposta correta. Assim, no modelo de Dahbura, Sabnani e King, a probabilidade da comparação resultar em igualdade para a comparação de duas unidades falhas é q^2.

No modelo de Pelc um sistema é chamado de diagnosticável se para qualquer síndrome possível, existe um único conjunto mais provável de unidades falhas gerando esta síndrome. Se este conjunto existe, ele é diagnosticado como as unidades falhas do sistema. Considerando o modelo (p, k)-probabilístico, os autores provam que:

1. Um sistema com duas unidades ($N = 2$) não é diagnosticável.
2. Assumindo que $p < 1/(k + 1)$, um sistema ótimo diagnosticável com $N > 2$ unidades possui $N - [N/3]$ arestas ou *links* de conexão.
3. Os problemas do *diagnóstico* e da *diagnosticabilidade* são NP-difíceis (*NP-hard*), neste modelo, para sistemas de topologias arbitrárias.

Blough e Pelc em [23] apresentam um algoritmo polinomial de diagnóstico para o modelo de Pelc [156], considerando uma grande classe de sistemas representados por grafos bipartidos, que incluem hipercubos, grades e florestas. Eles também mostram que o diagnóstico ótimo para sistemas de topologia geral é NP-difícil. Um algoritmo de tempo linear para realizar o diagnóstico ótimo em um anel também é apresentado.

Outro modelo probabilístico baseado em comparações é apresentado por Rangarajan e Fussel em [163] e é baseado na avaliação de múltiplas síndromes, ao invés de apenas uma. Em [89] os mesmos autores propõem um algoritmo para este modelo, no qual a probabilidade do diagnóstico correto se aproxima de 1 quando o número de testes realizados em cada processador é ligeiramente maior que $log_2 N$. Em [128] um algoritmo ótimo para o mesmo modelo é apresentado. Uma solução para o diagnóstico probabilístico de sistemas esparsamente interconectados é apresentada em [38].

A.12 Diagnóstico Evolucionário Baseado em Comparações

O diagnóstico evolucionário em nível de sistema foi introduzido por Elhadef e Ayeb em [67]. Aquele trabalho teórico investiga como um Algoritmo Genético (*Genetic Algorithm* - GA) é executado quando aplicado ao problema da identificação de unidades falhas a partir de uma síndrome, considerando o modelo PMC. Vários outros algoritmos evolucionários também foram implementados e comparados em [152]. O diagnóstico baseado em comparações baseado em computação evolucionária também foi apresentado por Elhadef e Ayeb em [68].

Um algoritmo genético possui os seguintes componentes [68]:

1. Uma representação de potenciais soluções para o problema, chamada de *cromossomo* ou *indivíduo*. Cada cromossomo é um *array* binário de tamanho N que quando usado para o diagnóstico do sistema representa quais nodos estão falhos e quais estão sem-falha. O cromossomo é representado por $\langle s_1 s_2 s_3 \ldots s_N \rangle$ onde s_i é o estado do nodo $u_i \in V$. O estado s_i do nodo u_i – também chamado de *gene* – pode ser 0 (sem-

falha) ou 1 (falho). Por exemplo, para um sistema de 8 nodos, o cromossomo $v = \langle 01000100 \rangle$ representa uma potencial solução onde o nodo 2 e o nodo 6 são falhos. Um conjunto de indivíduos é chamado de uma *população*.

2. Um procedimento para criar uma população inicial de soluções.

3. Uma função de avaliação que indica a *aptidão* (*fitness*) de cada indivíduo. A função de avaliação pode ser vista como a probabilidade de uma potencial solução estar correta.

4. Operadores genéticos, que são empregados para modificar indivíduos de uma população para produzir novos indivíduos. Operadores genéticos incluem, por exemplo, seleção, *crossover* e mutação, definidos como segue. A seleção forma uma nova geração através da escolha dos indivíduos da população anterior que possuem a maior aptidão. O *crossover* considera dois indivíduos – chamados pais – e produz novos indivíduos – chamados filhos – que por sua vez herdam materiais genéticos – *bits* – dos seus pais. A mutação troca *bits* aleatórios de indivíduos de uma população.

5. Parâmetros empregados pelos algoritmos genéticos, como o tamanho da população P e as probabilidades de aplicação dos operadores genéticos.

Cada um destes componentes possui impacto direto na solução obtida assim como no desempenho dos algoritmos genéticos. Elhadef e Ayeb apresentam um algoritmo genético – chamado *Genetic-Comparison-Diagnosis* – para o diagnóstico de falhas em sistemas sobre o modelo baseado em comparações. O algoritmo é apresentado na Figura A.22. O algoritmo recebe como entrada um grafo $G = (V, E)$ e a síndrome de comparações σ e produz como saída o conjunto dos nodos falhos F e o conjunto dos nodos sem-falha FF.

A função de aptidão de um cromossomo v, $FT(v)$, é apresentada abaixo. Algumas definições são necessárias para entendê-la. Seja $N(u_i)$ o conjunto de vizinhos do nodo u_i. Considerando o multigrafo $M = (V, C)$, $S_\sigma(u_i) = \{r((u_i, u_j)_{u_k}) \in \sigma$ tal que $u_j \in N(u_i)$ e $(u_i, u_j)_{u_k} \in C\}$. Em outras palavras, $S_\sigma(u_i)$ é o subconjunto de síndromes σ

```
Algoritmo: Genetic-Comparison-Diagnosis
/* Entrada:  Um grafo G = (V, E) e uma síndrome σ */
/* Saída:  Conjuntos de nodos Falhas F e sem-falha FF */

início
   Gerar população inicial de soluções Pop;
   para cada v ∈ Pop faça
         calcular FT(v);
   fim para
   Elite ← a solução Pop com a maior fitness;
   enquanto (∀v ∈ Pop, FT(v) ≠ 1) faça
         Selection(Pop);
         Mutation(Pop);
         Crossover(Pop);
         para cada v ∈ Pop faça
              calcular FT(v);
         fim para
         Elitism(Pop, Elite);
   fim enquanto
   F ← F(v) tal que v ∈ Pop e FT(v) = 1;
   FF ← V − F;
fim
```

Figura A.22: O algoritmo *Genetic-Comparison-Diagnosis*.

correspondente às comparações entre ao nodo u_i e seus vizinhos $N(u_i)$. Considerando o cromossomo v, $v[i]$ denota o i-ésimo *bit* de um *array* binário v, e σ^* denota a síndrome de comparações correspondente. A valor de aptidão de um nodo u_i é dado por $f(v[i])$, isto é, $f(v[i])$ é a probabilidade do estado do nodo u_i estar correto.

$$FT(v) = \frac{\Sigma_{i=1}^{n} f(v[i])}{N}, \text{ onde } f(v[i]) = \frac{|S_\sigma(u_i) \cap S_{\sigma^*}(u_i)|}{|N(u_i)|}.$$

Este algoritmo genético possui uma pequena modificação comparado aos GAs tradicionais [68]: o processo de mutação é realizado antes do *crossover*. Esta ordem é empregada pois o operador de mutação usado no algoritmo *Genetic-Comparison-Diagnosis* é baseado no valor de aptidão. Nos processos de mutação tradicionais, cada *bit* possui uma chance igual de sofrer mutação. Ao invés disso, os autores consideram cada *bit* do valor de aptidão $f(v[i])$ como a sua probabilidade de ser trocado. Assim, cromossomos não devem sofrer *crossover* antes de mutação. Os autores apresentam resultados experimentais comparando o operador padrão de mutação (chances iguais) com o novo processo de mutação em sistemas com a quantidade de nodos variando de 8 a 500. Os resultados mostram que

o algoritmo sobre o novo operador de mutação completa o diagnóstico em menos gerações.

O algoritmo *Genetic-Comparison-Diagnosis* usa a estratégia de elitismo (*elitism*), isto é, ao final de cada iteração, o melhor cromossomo é sempre comparado com um cromossomo de elite – que é o melhor cromossomo até aquele momento, e existe uma cópia deste cromossomo armazenada separadamente da população. Se o melhor cromossomo é melhor que o cromossomo de elite, uma cópia dele vira o cromossomo de elite. Por outro lado, se o melhor cromossomo não é melhor que o de elite, uma cópia do cromossomo de elite substitui o pior cromossomo na população. O elitismo garante que a qualidade da melhor solução encontrada com o passar de gerações é sempre maior.

Finalmente, $\mathcal{F}(v)$ denota o conjunto de nodos falhos de acordo com o cromossomo v, que são os nodos cujos genes possui valor igual a 1. Na função de aptidão FT, se o cromossomo v corresponde à solução ótima, ou seja, $\mathcal{F}(v)$ é o conjunto de todos os nodos falhos no sistema, então $FT(v) = 1$ e v é o diagnóstico do sistema.

A ordem de complexidade do algoritmo *Genetic-Comparison-Diagnosis* é $O((|E|\, P \ln P^2)/\ln r)$ no pior caso e $O((|E|\, P \ln P)/\ln r)$ na média, onde P é o tamanho da população e r é a taxa de aptidão. Elhadef e Ayeb também propuseram em [70, 69] outro algoritmo de diagnóstico baseado em comparações, incluindo um algoritmo genético serial. Abrougui e Elhadef em [1] apresentam uma versão paralela do modelo de diagnóstico evolucionário existente, e também apresentam um algoritmo de diagnóstico genético e paralelo.

A.12.1 Sistemas Imunológicos e Redes Neurais Artificiais Utilizando Diagnóstico Baseado em Comparação

Um sistema imunológico artificial (AIS - *Artificial Immune System*) é projetado para imitar as operações do sistema imunológico humano. O projeto de um AIS é bem similar ao projeto de outras abordagens de inteligência computacional tradicionais, como os algoritmos genéticos. Estes sistemas têm sido utilizados em várias aplicações, incluindo o diagnóstico clássico em nível de sistemas [7, 49, 112].

Elhadef, Das e Nayak [74] argumentam que os algoritmos de diagnóstico genético sofrem uma perda de diversidade da população, especialmente devido ao uso de um operador adaptativo de mutação. Esta característica causa um tempo de execução muito grande no pior caso, quando comparado ao caso médio. Em [74] Os autores resolveram este problema sobre o modelo de diagnóstico baseado em comparações apresentando uma abordagem baseada em sistemas imunológicos artificiais, que preserva a diversidade da população evitando o pior caso dos algoritmos genéticos.

Em [64] Elhadef utiliza uma rede neural artificial para resolver o problema do diagnóstico baseado em comparações. Uma rede neural artificial (*artificial neural network*, ou ANN) pode ser definida como um modelo de raciocínio (ou aprendizado) que se baseia no cérebro humano. Uma ANN é formada por um conjunto de neurônios artificiais interconectados. As interconexões entre neurônios artificiais também podem ser chamadas de sinapses, e possuem cada, uma um peso (ou parâmetro) ajustável. Um neurônio recebe sinais de entrada e transmite sinais, através da sua conexão de saída, para outros neurônios da rede. Uma rede neural artificial, pode ser considerada um sistema não linear e paralelo para aprendizado adaptativo e processamento de informações. As redes neurais podem ser usadas para modelar relacionamentos complexos entre entradas e saídas, ou para a identificação de padrões de dados. Em outras palavras, as ANNs são sistemas que permitem resolver problemas de classificação, reconhecimento de padrões, tomada de decisões, entre outros.

Elhadef utiliza uma rede neural *perceptron-based* [64] para realizar o diagnóstico de sistemas com base no modelo apresentado por Chwa e Hakimi [42]. O autor enfatiza que uma rede *perceptron-based* é a forma mais simples de uma rede neural artificial, que consiste de um neurônio, e um conjunto de pesos ajustáveis, que por sua vez são utilizados para modelar os testes (ou comparações). O propósito de uma rede *perceptron-based* é classificar entradas – como a síndrome – em uma de duas classes – neste caso, o conjunto das unidades falhas e sem-falha. É importante lembrar que no modelo de diagnóstico baseado em comparações de Chwa e Hakimi, a comparação de tarefas executadas por

pares de unidades falhas pode resultar em igualdade. Entretanto, os autores concluem que, quando aplicada ao modelo baseado em comparações de Chwa e Hakimi, o algoritmo *perceptron-based* não foi capaz de diagnosticar corretamente as unidades falhas dos sistemas em todos os casos.

Já em [75] Elhadef e Nayak, também aplicam o modelo baseado em comparações de Chwa e Hakimi, mas para BPNNs (*Backpropagation Neural Networks*), e apresentam um novo algoritmo de diagnóstico. Uma *backpropagation neural network* é uma rede neural artificial composta de múltiplas camadas de neurônios. Além disso, uma BPNN possui duas fases de aprendizado: primeiramente uma determinada entrada de aprendizado é apresentada à rede e sua saída é observada; se a saída for diferente da esperada, um erro é calculado e propagado de volta através da própria rede, para permitir o ajuste de parâmetros. Os autores apontam que a fase de aprendizado do algoritmo em BPNNs é *off-line*, e que então não possui impacto na latência do diagnóstico. Os autores concluem que, quando comparado a outros algoritmos evolucionários, o algoritmo proposto completa o diagnóstico de forma mais eficiente, mesmo em sistemas maiores.

Em [76] Elhadef e Nayak também apresentam um algoritmo aplicado a redes neurais, mas considerando o modelo generalizado de diagnóstico baseado em comparações apresentado por Sengupta e Dahbura [169]. Este modelo é uma extensão do modelo MM onde as unidades comparadoras pode ser ao mesmo tempo uma das unidades comparadas. O algoritmo apresentado também é baseado em BPNNs e também explora a fase de aprendizado *off-line* da rede neural para tornar o algoritmo de diagnóstico mais eficiente. Entre os resultados, os autores mostram que o algoritmo realizou corretamente o diagnóstico do sistema em cerca de 97% a 98% das configurações de rede simuladas.

Já em [65] os autores apresentam uma solução aplicada a redes neurais *Hopfield* também com base no modelo generalizado de Sengupta e Dahbura. Uma das principais características de uma rede neural *Hopfield* é que ela assume que todos os neurônios são completamente interconectados. Os autores enfatizam que o algoritmo neural proposto é capaz de realizar o diagnóstico das unidades falhas na presença de síndromes

parciais, ou seja, mesmo quando parte dos resultados de comparações não está disponível. Os resultados apresentados mostram que a estratégia foi capaz de realizar corretamente o diagnóstico em um número maior de casos, quando comparada ao algoritmo aplicado em BPNNs. Por fim, os autores de [77] estendem estes trabalhos e realizam novos experimentos com algoritmos neurais aplicados a BPNNs, com ambos os modelos de Chwa e Hakimi e de Sengupta e Dahbura. Ambos os modelos são também avaliados na presença de síndromes parciais. Os resultados mostram que, para algumas configurações de sistemas simulados, o algoritmo foi capaz de identificar corretamente as unidades falhas do sistema em cerca de 99% dos experimentos, nos casos onde no máximo até a metade dos resultados de comparações estavam ausentes.

Recentemente em [66] Elhadef investiga o diagnóstico baseado em comparações usando SVMs lineares (*Linear Support Vector Machines*) [28, 182], com base no modelo inicial de diagnóstico baseado em comparações proposto por Malek [143]. Os SVMs são considerados um método estático e robusto de aprendizado, que tem o objetivo de resolver problemas que também incluem o de classificação e reconhecimento de padrões. Elhadef então apresenta que o diagnóstico sobre os modelos baseados em comparações pode ser descritos como: (a) um problema de classificação, no qual o objetivo é classificar os nodos do sistema como falhos e sem-falha; e (b) um problema de reconhecimento de padrões [28], onde o objetivo é organizar dados de entrada – como uma síndrome – em categorias – como o conjunto correspondente de unidades falhas.

Um algoritmo de diagnóstico é então proposto [66], baseado em SVMs, e que utiliza a fase de aprendizado com base em diversas síndromes onde os respectivos conjuntos de unidades falhas são conhecidos. Resultados realizados através de simulações mostram que mesmo na presença de síndromes parciais, em algumas configurações particulares o algoritmo foi capaz de realizar corretamente o diagnóstico das unidades falhas em cerca de 99% dos casos simulados – nos experimentos onde no máximo até a metade dos resultados de comparações estavam ausentes. Por outro lado, os autores concluem, também com base em experimentos realizados, que o algoritmo baseado em SVMs não realiza corretamente

o diagnóstico com uma alta porcentagem de acerto, quando aplicado ao modelo de Chwa e Hakimi [42] – modelo no qual a comparação de saídas de duas unidades falhas pode resultar em igualdade.

A.13 Diagnóstico Baseado em Comparações Aplicado a Redes Ad Hoc

As redes móveis ad hoc (*mobile ad hoc networks* - MANETs) implementam um ambiente cooperativo distribuído, que consiste de diferentes dispositivos móveis e sem fio (chamados nodos), que variam de pequenos dispositivos de mão a computadores portáteis. Estas redes são baseadas no paradigma *peer-to-peer*. Como existe um alcance limitado para a comunicação sem fio, a rede é geralmente *multihop*, pois comunicação direta entre os nodos geralmente não está disponível. A comunicação de um nodo se dá através da transmissão de mensagens a outros nodos que estão no seu raio de transmissão. Um problema maior surge com a mobilidade dos nodos, o que causa variação na topologia da rede e, em certa medida, é imprevisível. De fato, os enlaces de comunicação entre os nodos pode quebrar, nodos podem falhar, outros nodos podem se recuperar de falhas, e ainda novos enlaces podem aparecer [16, 101, 3]. Além disso, os nodos confiam em baterias como suplemento de energia e o efeito do esgotamento de baterias é similar a uma falha *crash*, onde o nodo deixa de funcionar.

O diagnóstico baseado em comparações foi aplicado para redes móveis ad hoc por Chessa e Santi em [34] e Elhadef, Boukerche e Elkadiki em [71, 73]. Protocolos para ambos os modelos apresentados em ambos os trabalhos são capazes de identificar falhas do tipo *hard* (permanentes) e *soft* (temporárias). Um nodo sofre uma falha do tipo *hard* quando ele cessa completamente a comunicação com os outros nodos. Por outro lado um nodo afetado por uma falha *soft* continua a operar e comunicar, mas com comportamento alterado. A descrição de ambos os modelos segue abaixo.

A.13.1 Modelo de Diagnóstico de Chessa e Santi

O modelo proposto por Chessa e Santi [34] é baseado no modelo MM*. Os autores apresentam duas implementações deste modelo. Na primeira, a topologia da rede não muda durante o diagnóstico. Já na segunda, a topologia da rede pode variar durante o decorrer do diagnóstico.

A topologia do sistema no tempo τ é modelada como um grafo direcionado $G(\tau) = (V, L(\tau))$, onde V é o conjunto de nodos e $L(\tau)$ é o conjunto dos enlaces lógicos existentes no tempo τ. Dados quaisquer dois nodos $u, v \in V$, existe uma aresta $(u \rightarrow v \in L(\tau))$ se e somente se v está no raio de transmissão de u no tempo τ. O modelo consiste somente de enlaces bidirecionais, assim se $(u \rightarrow v \in L(\tau))$ então $(v \rightarrow u \in L(\tau))$, e $G_\tau = (V, L(\tau))$ é não direcionado. O conjunto de nodos dentro do alcance de transmissão de um dado nodo u no tempo τ é chamado de conjunto de vizinhos de u no tempo τ, denotado por $N(u, \tau)$ ou simplesmente $N(u)$.

Este modelo faz as seguintes asserções:

1. Cada nodo possui um identificador único;
2. Existe um protocolo em nível de enlace que provê o seguinte:
   (a) Resolve contenções;
   (b) Provê a primitiva de *broadcast* confiável *one-hop*, chamado $1\_rb(\cdot)$;
   (c) Identifica a nodo de origem de uma mensagem recebida.

As comparações entre unidades tiram proveito da natureza compartilhada do enlace de comunicação. Uma unidade testadora sem-falha j envia como teste requisições aos seus vizinhos e espera pelas suas respostas. Assim que as respostas forem recebidas, as unidades são diagnosticadas com base nas asserções do modelo MM*.

Dependendo da topologia da rede, diferentes decisões sobre o estado (falho ou sem-falha) das unidades que não responderam à requisição enviada como teste podem ser tiradas, e são descritas a seguir.

A.13.1.1 Protocolo de Comparações para Topologia Fixa

Este protocolo, também chamado de *Static Distributed Self-Diagnosis Protocol* (*Static-DSDP*), assume que a topologia da rede não muda durante a execução dos testes, isto é, se uma unidade u envia uma requisição de teste no tempo τ, e T_{out} é o tempo limite para este teste ser executado (*timeout*), então $N(u, \tau') = N(u, \tau) = N(u)$ para todo $\tau < \tau' \leq \tau + T_{out}$. Esta asserção não quer dizer que a rede é estática, mas sim que sua topologia não muda durante o diagnóstico; em outras palavras, os nodos podem se movimentar, mas eles não pode migrar para fora do alcance de transmissão dos seus vizinhos.

As comparações são realizadas com base no seguinte protocolo:

— *Geração da requisição de teste*: no tempo τ, a unidade u gera um número sequencial de testes i, uma tarefa T_i, o resultado esperado $R_{u,i}$ e envia a mensagem $m = (u, i, T_i)$ ao $N(u, \tau)$ usando a primitiva $1\_rb(m)$.

— *Recepção da requisição de teste*: toda unidade $v \in N(u)$, assim que recebe m, gera o resultado $R_{v,i}$ para T_i e invoca $1\_rb(m')$ no tempo τ', com $\tau < \tau' \leq \tau + T_{out}$. A mensagem $m' = (u, i, R_{v,i})$ é a resposta ao teste, e (u, i) é o cabeçalho da resposta.

— *Recepção da resposta do teste*: toda unidade $w \in N(v)$ recebe m'. Como $u \in N(v)$, u também recebe m', e compara $R_{u,i}$ e $R_{v,i}$: v é diagnosticado como sem-falha se o resultado é 0, e como falho no caso contrário. Para $w \in N(v)$ mas $w \neq u$, ou $w \in N(u)$ ou $w \notin N(u)$. Caso $w \in N(u)$, como mostrado na Figura A.23(a), w compara $R_{v,i}$ e $R_{w,i}$: v é diagnosticado como sem-falha se o resultado da comparação é 0, e como falho no caso contrário. Caso $w \notin N(u)$, como mostrado na Figura A.23(b), se w também tiver recebido outro mensagem m'' sobre a mesma tarefa a partir do nodo $z \in N(u)$, w compara $R_{z,i}$ e $R_{v,i}$. Se a comparação indicar igualdade, então os nodos v e z são diagnosticados como sem-falha. Se o resultado da comparação indicar diferença, e se z já tiver sido diagnosticado como sem-falha, então v é diagnosticado

como falho. Finalmente caso $w \notin N(u)$ e w não tiver recebido outra mensagem sobre a mesma tarefa, então $R_{v,i}$ é apenas armazenado localmente.

— *Limite de tempo* (*timeout*): No tempo $\tau + T_{out}$ o nodo u diagnostica como falhos todos os outros nodos que não responderam à requisição de teste.

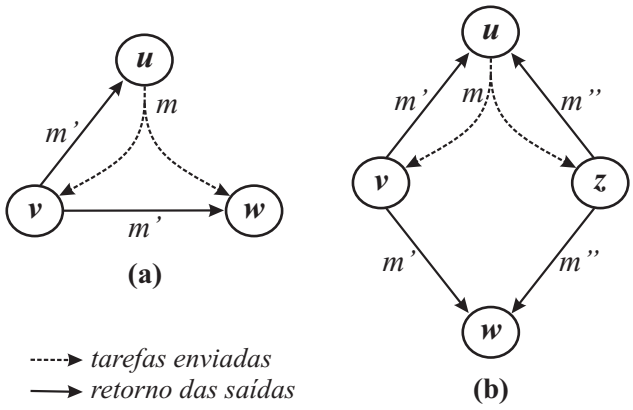


Figura A.23: (a) A unidade w recebeu a requisição de teste m da unidade u. (b) A unidade w recebe a resposta m' e m'' referente à requisição de teste m.

Os autores mostram que, assumindo uma rede de topologia fixa, se um nodo sem-falha u gera uma requisição de teste no tempo τ, então no tempo $\tau + T_{out}$:

— A unidade u diagnosticou corretamente o estado de todas as unidades em $N(u)$.

— Toda unidade sem-falha $v \in N(u)$ diagnosticou corretamente o estado de unidades sem-falha e unidades com falha do tipo *soft* em $N(u) \cap N(v)$.

— Toda unidade sem-falha $z \in N_2(u)$, onde $N_2(u) = \{z \in V - N(u)$ tal que $|N(u) \cap N(z)| \geq 2\}$, diagnosticou corretamente o estado de todas as unidades sem-falha e unidades com falha do tipo *soft* em $N(u) \cap N(z)$ se ao menos duas unidades em $N(u) \cap N(z)$ estão sem-falha.

A.13.1.2 Protocolo de Comparações para Topologia Variante no Tempo

Assuma agora que os nodos podem migrar durante a execução dos testes. As comparações são realizadas de acordo com a seguinte dinâmica, isto é, protocolo de topologia variante no tempo:

— *Geração da requisição de teste*: no tempo τ, a unidade testadora u gera um número sequencial de testes i, uma tarefa T_i, o resultado esperado $R_{u,i}$ e envia a mensagem $m = (u, i, T_i)$ para $N(u, \tau)$ usando $1\_rb(m)$.

— *Recepção da requisição de teste*: toda unidade $v \in N(u, \tau)$, assim que recebe m, gera a resposta $R_{v,i}$ para T_i e invoca $1\_rb(m')$ no tempo τ', com $\tau < \tau' \leq \tau + T_{out}$, onde $m' = (u, i, R_{v,i})$.

— *Recepção da resposta do teste*: toda unidade $w \in N(v, \tau)$, assim que recebe m', faz o seguinte: se $w = u$, ela compara $R_{v,i}$ com o resultado esperado $R_{u,i}$ e gera o resultado da comparação. A unidade v é diagnosticada como sem-falha se o resultado é 0, e como falha no caso contrário. Se $w \neq u$, os seguintes casos ocorrem: (a) $w \in N(u, \tau)$. Neste caso, a unidade w recebe a requisição de teste m de u, portanto ela pode comparar $R_{v,i}$ com $R_{w,i}$. A unidade v é diagnosticada como sem-falha se o resultado da comparação é 0, e como falha no caso contrário. (b) $w \notin N(u, \tau)$. A unidade v não está com falha do tipo *hard*, e sua resposta de teste é comparada com as respostas recebidas para o mesmo teste, se existir alguma. Se existir algum $z \in N(u)$ tal que $R_{z,i} = R_{v,i}$ então ambas as unidades são diagnosticadas como sem-falha; caso contrário, se a unidade z já tiver sido diagnosticada como sem-falha, então v é diagnosticada como falha. Caso contrário, a resposta do teste $R_{v,i}$ é armazenada.

— *Limite de tempo* (*timeout*): No tempo $\tau + T_{out}$ o nodo u diagnostica como falhos todos os outros nodos que não responderam à requisição de teste.

Como a topologia da rede varia com o tempo, em geral $N(u,\tau) \neq N(u,\tau + T_{out})$. Como uma consequência, unidades com falha do tipo *hard* não podem ser diferenciadas de unidades sem-falha que migraram para fora do alcance de transmissão das unidades testadas. Por este motivo, o testador consegue apenas classificar as unidades que não responderam à requisição de teste.

Os autores mostram que se um nodo sem-falha u gera uma requisição de teste no tempo τ, e a topologia da rede pode mudar durante o diagnóstico, então, no tempo $\tau + T_{out}$, o nodo u terá diagnosticado corretamente o estado de todos os nodos sem-falha e os nodos com falha *soft* em $N(u,\tau) \cap N(u,\tau + T_{out})$.

A.13.2 Modelo de Diagnóstico de Elhadef, Boukerche e Elkadiki

Em [73, 71] Elhadef, Boukerche e Elkadiki apresentam protocolos de diagnóstico baseado em comparações para redes móveis ad hoc. Dois protocolos são apresentados: o *Adaptive Distributed Self-Diagnosis Protocol* (*Adaptive-DSDP*) para redes de topologia fixa, e o *Mobile Distributed Self-Diagnosis Protocol* (*Mobile-DSDP*) para redes de topologia variante no tempo. A ideia chave de ambos os protocolos é que um nodo, quando responde a uma requisição de teste, deve também enviar a tarefa recebida como teste juntamente com a sua saída para aquela tarefa. Assim qualquer nodo que as receber, terá condições de diagnosticar o estado daquele nodo testado através da comparação da saída recebida com alguma outra saída já recebida para a mesma tarefa enviada como teste, ou mesmo através da comparação da saída recebida com a sua própria saída após executar a mesma tarefa de teste.

Além do fato de que os nodos retransmitem a tarefa junto com as resposta dos testes, o modelo de diagnóstico de topologia fixa no qual o *Adaptive-DSDP* é baseado também diferencia-se do modelo de Chessa e Santi [34] na estratégia de disseminação. No modelo de Chessa e Santi, assim que um nodo coleta todas as respostas dos seus vizinhos, ele transmite a sua visão local de todos os nodos na MANET usando uma fase de disseminação baseada em inundação (*flooding*). Por outro lado, o *Adaptive-DSDP* usa uma árvore

geradora mínima (*spanning tree*) em uma estratégia de disseminação epidêmica (*gossip*) [73].

Este novo protocolo de comparações com topologia variante no tempo é descrito a seguir. Neste protocolo, os vizinhos de um nodo são classificados com estáveis ou dinâmicos. Vizinhos dinâmicos são os que acabaram de se mover para a vizinhança de um determinado nodo.

— *Geração da requisição de teste*: um nodo u transmite a requisição de teste para os seus vizinhos em um dado tempo τ. A requisição de teste inclui a tarefa, T_i, onde i é um número sequencial que identifica o teste. Após enviar a requisição de teste, $\langle Test, T_u \rangle$, um temporizador (*timer*) é definido com T_{out}. Além disso, um segundo temporizador é definido com $T_{DiagnosisSession}$, que refere-se ao pior caso da latência de diagnóstico se todos os nodos são sem-falha. Este segundo temporizador é usado para identificar nodos dinâmicos com falhas *hard* e que não respondem às requisições de teste, ou que podem ter se movido o suficiente para que seus estados não sejam diagnosticados pelos outros nodos.

— *Recepção da requisição de teste*: quando um nodo v recebe uma requisição de teste de um de seus vizinhos u, o nodo v a trata da seguinte forma. Se ele já sabe qual é a resposta R para a tarefa de teste T_u, então ele atribui $R_u^v = R$; caso contrário, ele executa a tarefa T_u e gera a resposta de saída R_u^v. Então ele transmite para todos os seus vizinhos a resposta do teste através da mensagem $\langle Response, T_u, R_u^v \rangle$, que contém a tarefa de testes T_u e sua resposta R_u^v. A resposta do teste é armazenada em um conjunto de respostas, denotado por $Validated_v$, no qual todas as respostas corretas de testes – que inclui as geradas pelo próprio nodo e também as deduzidas durante a sessão de diagnóstico – são mantidas. Nesta etapa, o nodo v gera a sua própria requisição de teste, se ainda não o fez, e a envia para todos os seus vizinhos. Cada nodo deve responder a no máximo $t + 1$ requisições de testes se o sistema for t-diagnosticável.

— *Recepção da resposta do teste*: quando se trata as respostas de testes, diferentes cenários devem ser considerados. O nodo w pode receber respostas de teste de seus vizinhos estáveis e dos dinâmicos. Quando os vizinhos são estáveis ou dinâmicos, se o nodo w receber a saída da tarefa destes nodos vizinhos juntamente com o conjunto de tarefas que eles executaram para gerar estas saídas, o nodo w irá ser capaz de diagnosticar os seus estados. Todas as respostas de testes recebidas por w no qual ele é incapaz de classificar como correta são armazenadas em um conjunto de nodos pendentes, chamado $Pending_w$.

— *Limite de tempo* (*timeout*): após a ocorrência do primeiro *timeout* T_{out}, o nodo u é capaz de diagnosticar o estado de seus vizinhos estáveis bem como os vizinhos dinâmicos dos quais ele já recebeu pelo menos uma resposta de teste. Nesta etapa, o nodo u dissemina sua visão local de diagnóstico para todos os seus vizinhos. Quando o segundo *timeout* ocorre, $T_{DiagnosisSession}$, o nodo u irá considerar todos os nodos remanescentes como falhos.

Elhadef, Boukerche e Elkadiki apresentam em [72] outro protocolo de diagnóstico distribuído baseado em comparações para redes móveis ad hoc baseado no modelo de Chessa e Santi. O protocolo proposto é chamado *Dynamic-DSDP* e também identifica falhas do tipo *hard* e *soft*.

Os autores comparam o protocolo *Dynamic-DSDP* com o protocolo *Static-DSDP* de Chessa e Santi. Considere as seguintes três definições. (1) T_{gen} é o limite máximo de tempo decorrido entre a recepção da primeira mensagem de diagnóstico e a geração da requisição de teste correspondente. (2) Uma mensagem de diagnóstico pode ser uma requisição de teste, uma resposta de teste, um *timeout* ou ainda a disseminação de uma mensagem. (3) T_f é um limite máximo para o tempo necessário para propagar a disseminação de uma mensagem. O protocolo *Dynamic-DSDP* possui ordem de complexidade $O(\Lambda(T_{gen}+d_{ST}T_f)+T_{out})$ enquanto que o protocolo *Static-DSDP* é $O(\Lambda(T_{gen}+T_f)+T_{out})$, onde Λ denota o diâmetro do grafo G e d_{ST} é a profundidade da árvore geradora mínima

usada para disseminar as mensagens. Além disso, o protocolo *Dynamic-DSDP* possui comunicação com complexidade $O(Nk_G) \simeq O(Nt)$ enquanto o *Static-DSDP* requer $O(N(N+1+\Delta)) \simeq O(N^2)$ mensagens, onde Δ é o grau do nodo de maior grau e k_G denota a conectividade de G.

Os autores também provam a complexidade do *Mobile-DSDP* em [71] e apresentam a análise do *Adaptive-DSDP* em [73]. O *Mobile-DSDP* possui ordem de complexidade $O(\hat{\Delta}(T_{gen}+T_f)+T_{out})$ e requer $O(N(N+\hat{k}))$ mensagens, onde $\hat{\Delta}$ e $\hat{k}$ denotam respectivamente o diâmetro máximo e a conectividade mínima do grafo G. O *Adaptive-DSDP* possui ordem de complexidade $O(\Lambda T_{gen} + (d_{ST}+N-1)T_f + T_{out})$ e sua comunicação possui ordem de complexidade $O(N\Delta)$.

A.14 Um Sumário dos Resultados do Diagnóstico em Nível de Sistema Baseado em Comparações

Esta seção sumariza os resultados relevantes do diagnóstico em nível de sistema baseado em comparações apresentados: a Figura A.24 sumariza os trabalhos do diagnóstico em nível de sistema apresentados, e as Figuras A.25, A.26 e A.27 sumariza os resultados relevantes do diagnóstico baseado em comparações. Em todos os grafos apresentados nas quatro figuras, um vértice representa um modelo, um algoritmo, ou então algum resultado relevante do diagnóstico em nível de sistema ou do diagnóstico baseado em comparações. Cada vértice possui dois rótulos: o rótulo interno lista os autores e o rótulo externo lista uma breve nota sobre a contribuição do respectivo trabalho. Uma aresta direcionada de um vértice a para um vértice b representa que o resultado identificado pelo vértice b é baseado em, é uma extensão do, ou ainda é relacionado ao resultado identificado no vértice a. Além disso, os grafos destas figuras estão ordenados em forma cronológica.

Os três grafos do diagnóstico baseado em comparações (Figuras A.25–A.27) são baseados no *survey* apresentado em [59], mas foram complementados com a inclusão dos novos trabalhos publicados na área a partir do ano de 2010 até a presente data. Estes três grafos ainda mostram o relacionamento entre os diversos resultados do diagnóstico baseado em comparações: quatro vértices aparecem nestes três grafos nomeados com [Malek 1980], [Chwa and Hakimi 1981], [Maeng and Malek 1981] e [Sengupta and Dahbura 1992] e aparecem em linhas pontilhadas. Estes quatro vértices representam as intercessões entre a cronologia apresentada nas três figuras. A Figura A.25 mostra os resultados derivados dos primeiros modelos de diagnóstico baseado em comparações – e que são baseados em ambos os modelos de Malek e de Chwa e Hakimi. A Figura A.26 mostra os resultados baseados no modelo MM; e, na sequência, a Figura A.27 mostra os resultados baseados no modelo MM*.

Por fim, as Tabelas A.1 e A.2 apresentam, respectivamente, um sumário mais detalhado de todos os resultados. A primeira tabela – Tabela A.1 – mostra um sumário dos

resultados do diagnóstico em nível de sistema, e a Tabela A.2 apresenta um sumário dos resultados do diagnóstico baseado em comparações. Todos os resultados estão agrupados pelo modelo no qual eles são baseados. As tabelas possuem três colunas. A primeira coluna indica o modelo de diagnóstico no qual o resultado se baseia. As próximas duas colunas apresentam respectivamente a referência para o trabalho e suas contribuições.

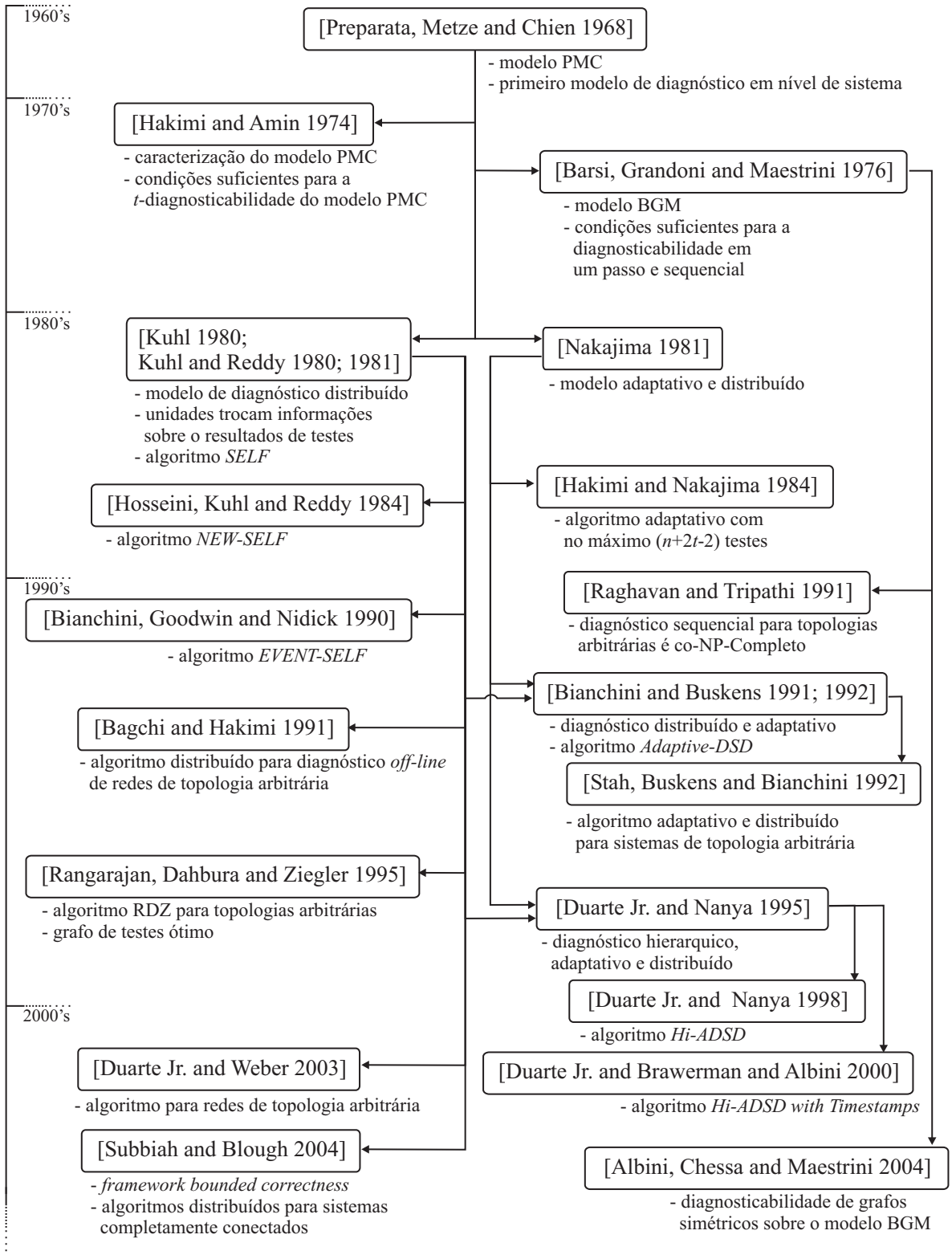


Figura A.24: Cronologia do diagnóstico em nível de sistema: resultados baseados nos trabalhos apresentados na Seção 2.1.

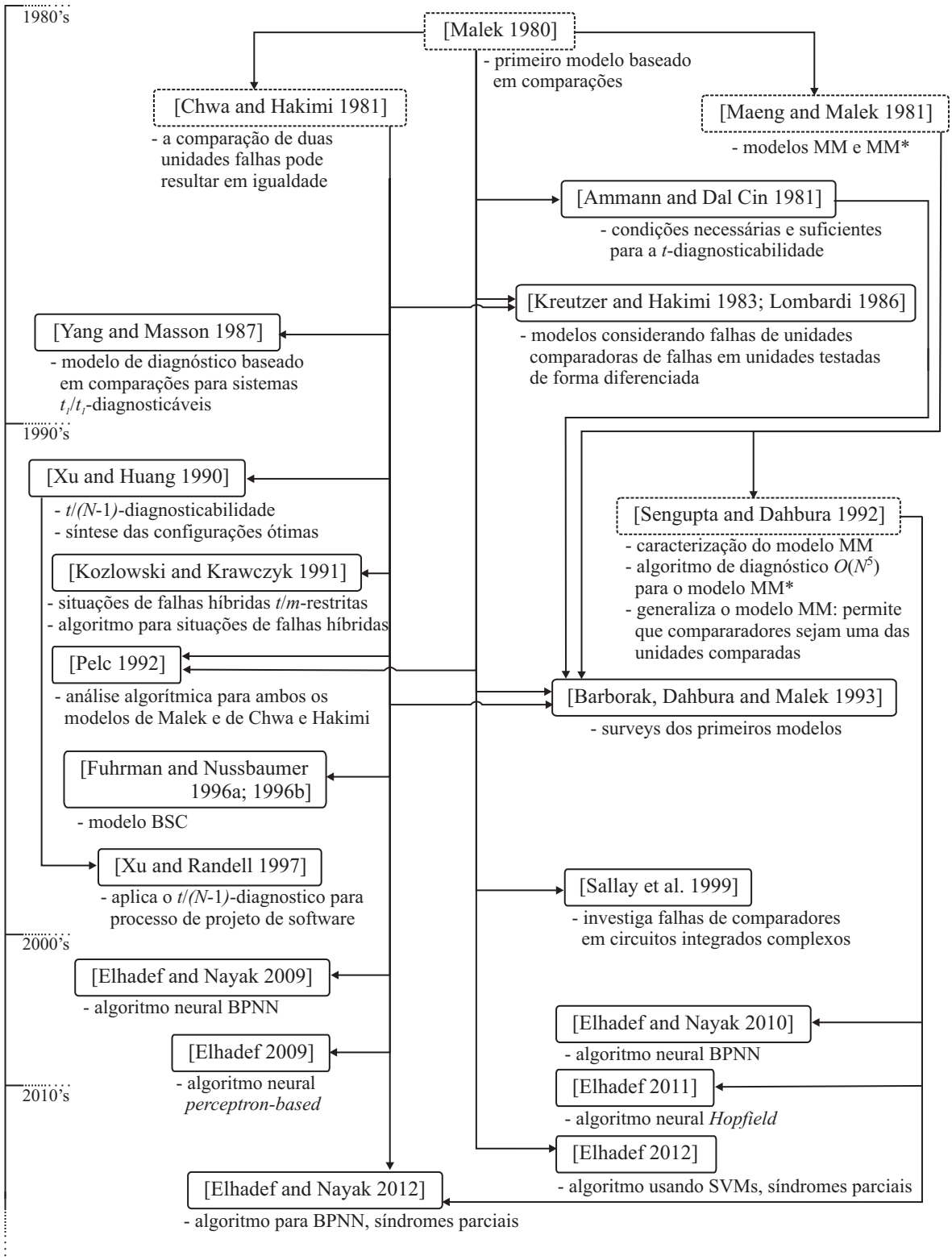


Figura A.25: Cronologia do diagnóstico baseado em comparações: resultados baseados nos primeiros modelos.

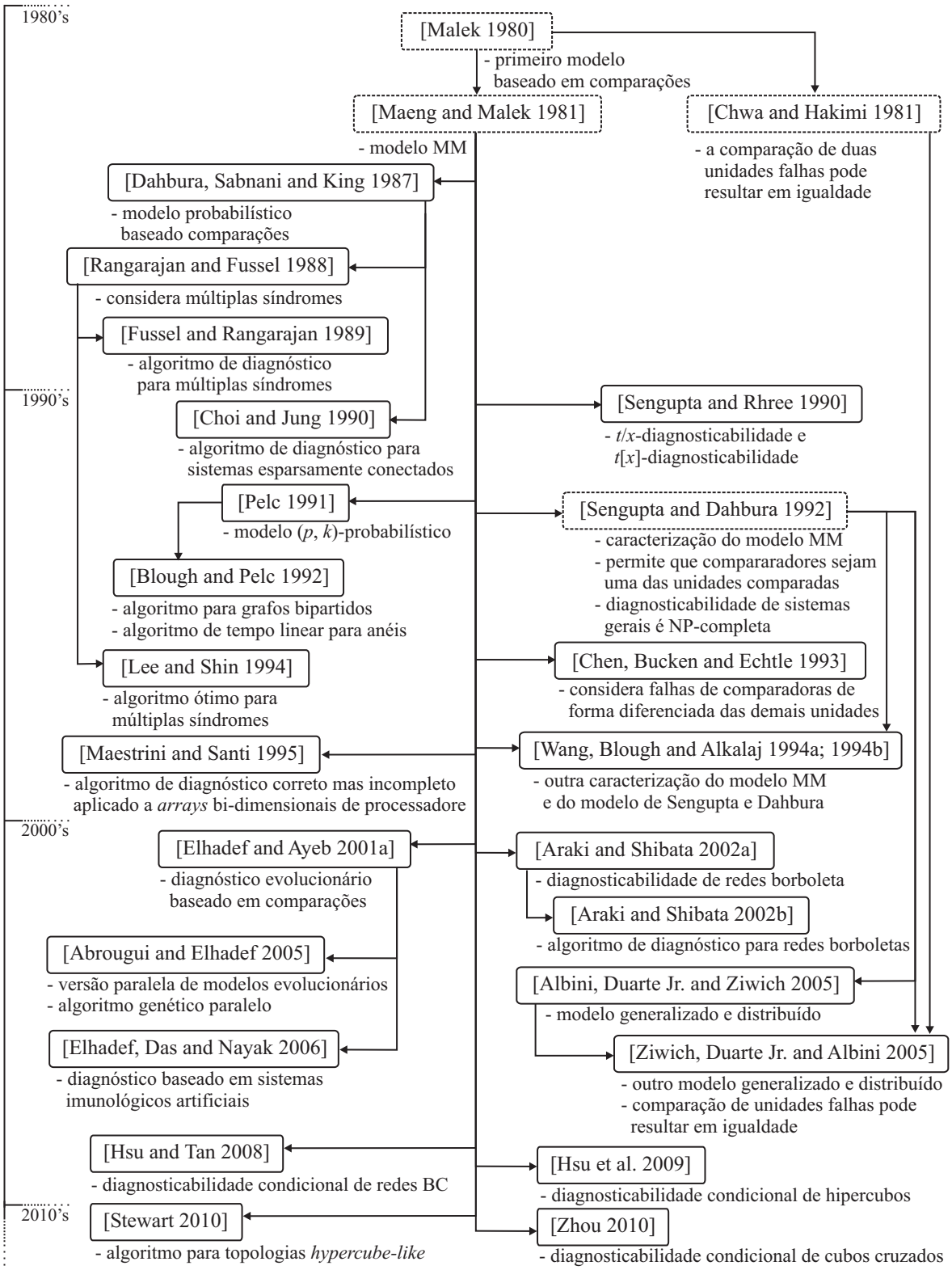


Figura A.26: Cronologia do diagnóstico baseado em comparações: resultados baseados no modelo MM.

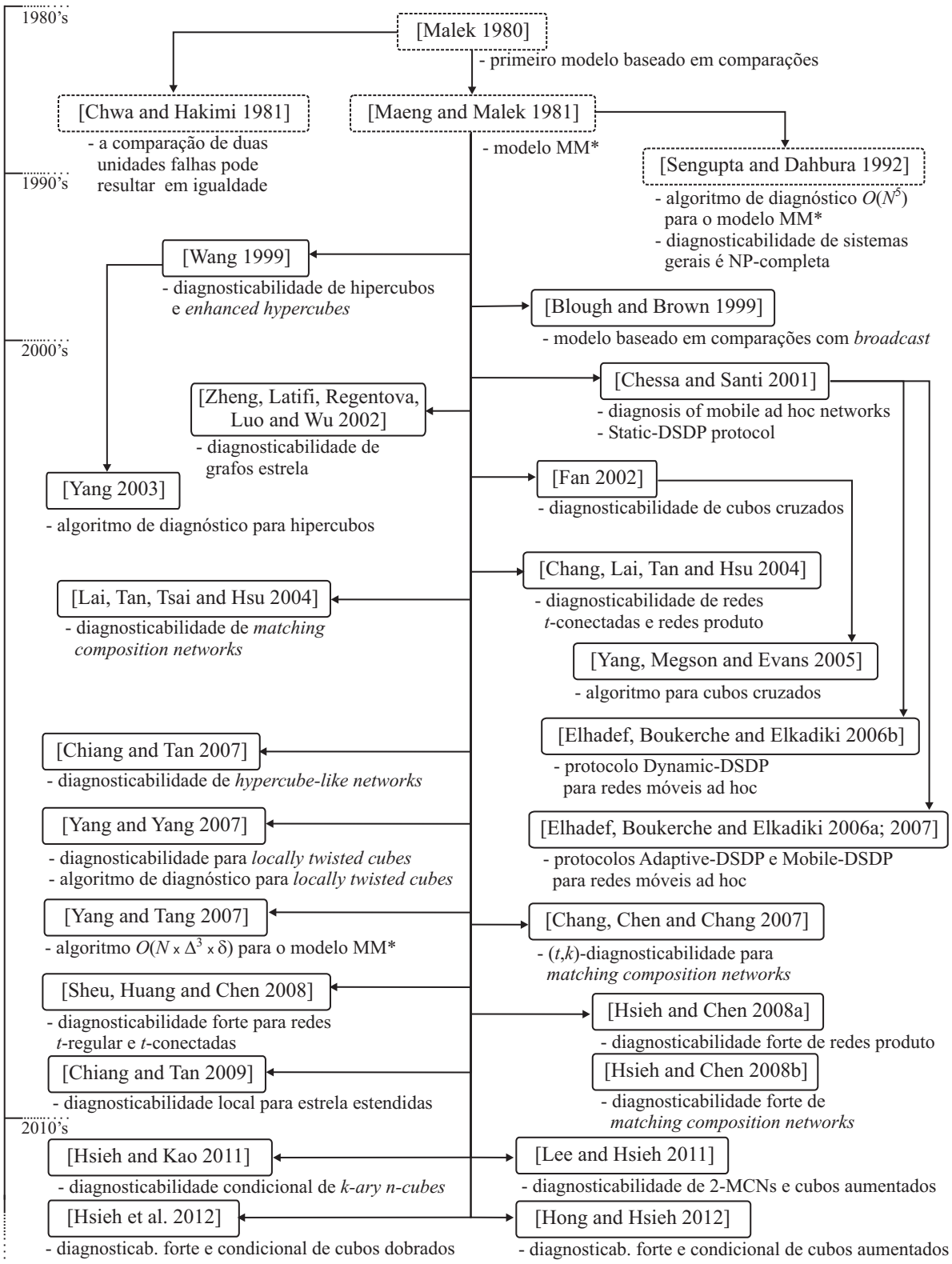


Figura A.27: Cronologia do diagnóstico baseado em comparações: resultados baseados no modelo MM*.

Modelo	**Referência**	**Principais Contribuições**
Modelo PMC	[158]	- modelo apresentado por Preparata, Metze e Chien - primeiro modelo de diagnóstico em nível de sistema - um teste envolve a aplicação controlada de estímulos e a observação da resposta correspondente
	[95]	- caracterização do modelo PMC - condições necessárias e suficientes para a t-diagnosticabilidade do modelo PMC
Modelo BGM	[15]	- outro modelo inicial para diagnóstico em nível de sistema proposto por Barsi, Grandoni e Maestrini - o teste de uma unidade falha sobre outra unidade falha deve resultar em *fail* - condições necessárias e suficientes para a t-diagnosticabilidade em um passo e sequencial
	[160]	- diagnóstico sequencial para topologias arbitrárias é co-NP-Completo
	[4]	- diagnosticabilidade de grafos simétricos
Diagnóstico Adaptativo	[151]	- introdução do primeiro modelo de diagnóstico adaptativo - testes são determinado de forma dinâmica - algoritmo adaptativo, com no máximo $(N-1)+t(t+1)$ testes
	[96]	- algoritmo com no máximo $(n+2t-2)$ testes
Diagnóstico Distribuído	[119, 120, 121]	- introdução do diagnóstico distribuído em nível de sistema - unidades sem-falha do sistema diagnosticam o estado de todas as unidades - unidades trocam informações sobre o resultados de testes - algoritmo *SELF*
	[103]	- algoritmo *NEW-SELF* de diagnóstico distribuído - permite a entrada de novas unidades ao sistema
	[21]	- algoritmo *EVENT-SELF* de diagnóstico distribuído
	[13]	- algoritmo *off-line* distribuído de diagnóstico para redes de topologia arbitrária
	[162]	- algoritmo *RDZ* para sistemas de topologia arbitrária - grafo de testes ótimo; cada nodo possui um testador
	[57]	- algoritmo para redes de topologia arbitrária - identifica quais partes da rede estão inalcançáveis
	[176]	- *framework* chamado *bounded correctness* - diagnóstico na presença de falhas e reparações dinâmicas - algoritmos distribuídos para sistemas completamente conectados e para redes de topologias arbitrárias
Diagnóstico Adaptativo e Distribuído	[19, 20]	- introdução do diagnóstico distribuído e adaptativo - algoritmo *Adaptive-DSD*
	[173]	- algoritmo adaptativo e distribuído para diagnóstico *on-line* de sistemas de topologia arbitrária
Diagnóstico Hierárquico, Adaptativo e Distribuído	[55]	- o diagnóstico hierárquico, adaptativo e distribuído em nível de sistema é apresentado - os nodos são agrupados em *clusters*
	[56]	- algoritmo hierárquico, adaptativo e distribuído *Hi-ADSD* - latência de no máximo $log_2^2 N$ rodadas de testes
	[54]	- algoritmo *Hi-ADSD with Timestamps* - constrói *clusters* sempre de tamanho $N/2$

Tabela A.1: Sumário do diagnóstico em nível de sistema: resultados apresentados na Seção 2.1.

Modelo	Referência	Principais Contribuições
Modelo apresentado por Malek	[143]	- primeiro modelo de diagnóstico baseado em comparações - unidades comparadas são diferentes - a comparação de uma ou duas unidades falhas resultam em diferença - observador central é uma unidade confiável que executa as comparações e realiza o diagnóstico - a diagnosticabilidade é $N-2$
	[8]	- condições necessárias e suficientes para a t-diagnosticabilidade
	[166]	- estratégia para identificar falhas em unidades comparadoras - aplicação para circuitos *wafer-scale*
	[157]	- análise algorítmica de ambos os modelos de Malek e de Chwa e Hakimi - pior caso do número de testes em algoritmos ótimos para o diagnóstico de t unidades falhas, diagnóstico sequencial e diagnóstico *one-step* para ambos os modelos, sobre as estratégias de teste adaptativa e não adaptativa
	[14]	- um *survey* dos primeiros modelos
Modelo apresentado por Chwa e Hakimi	[42]	- a comparação de duas unidades falhas pode resultar em igualdade
	[86, 85]	- modelo *Bounded Symmetric Comparison*, considera um limite no número de unidades falhas que podem produzir resultados idênticos
	[117]	- extensão do modelo apresentado por Chwa e Hakimi para situações de falhas hibridas t/m-restritas
	[194]	- modelo de diagnóstico t_1/t_1 baseado em comparações
	[192]	- caracterização da $t/(N-1)$-diagnosticabilidade sobre o modelo de Chwa e Hakimi - síntese das configurações ótimas $t/(N-1)$-diagnosticáveis para topologias como cadeias e *loops*
	[193]	- aplicação do $t/(N-1)$-diagnóstico para processos de projeto de software
	[118, 136]	- modelos considerando de forma separada as falhas de unidades comparadoras das falhas das outras unidades testadas - caracterização dos modelos propostos, $(t-t_c)$-diagnosticabilidade

Tabela A.2: Sumário dos resultados do diagnóstico baseado em comparações.

Modelo	Referência	Principais Contribuições
Modelo MM	[140]	- modelo de diagnóstico baseado em comparações no qual as unidades são também comparadoras - resultados de comparações quando pelo menos uma unidade é falha sempre resulta em diferença - observador central é uma unidade confiável que realiza o diagnóstico - condições necessárias e suficientes para a *one-step* t-diagnosticabilidade - procedimento para construir o grafo mínimo de sistemas diagnosticáveis - avaliação da latência de diagnóstico através de ciclos de testes
	[169]	- generalização do modelo MM: permite que comparadores sejam uma das unidades que são comparadas - caracterização de sistemas diagnosticáveis sobre o modelo MM - diagnosticabilidade de sistemas de topologia arbitrária é NP-completa
	[170]	- t/x-diagnosticabilidade e $t[x]$-diagnosticabilidade
	[33]	- extensão do modelo MM considerando falhas em compradores e nos demais processadores de forma separada; avaliação da diagnosticabilidade
	[187, 188]	- novas condições necessárias e suficientes para ambos o modelo MM e o modelo de Sengupta e Dahbura
	[141]	- algoritmo de diagnóstico correto, mas incompleto, aplicado para localizar falhas em *arrays* bidimensionais de processadores
	[10]	- diagnosticabilidade de redes borboletas k-aria r-dimensionais
	[11]	- algoritmo de diagnóstico $O(k^2n)$ para redes borboletas
	[109]	- diagnosticabilidade condicional de redes BC n-dimensionais X_n é $3(n-2)+1$ para $n \geq 5$
	[108]	- diagnosticabilidade condicional dos n-hipercubos é $3(n-2)+1$ para $n \geq 5$
	[174]	- algoritmo de diagnóstico $O(\Delta N)$ para topologias que abrangem os hipercubos, *enhanced hypercubes*, cubos cruzados, *twisted cubes*, grafos estrela, entre outros
	[205]	- diagnosticabilidade condicional de cubos cruzados CQ_n é $3n-5$ quando $n \geq 7$
Modelo MM*	[140]	- modelo MM* é um caso especial do modelo MM: cada unidade compara todo par de vizinhos
	[169]	- algoritmo de diagnóstico com ordem de complexidade $O(N^5)$ para o modelo MM* - diagnosticabilidade de sistemas de topologia arbitrária sobre o modelo MM* é NP-completa
	[198]	- algoritmo de diagnóstico de complexidade $O(N\Delta^3\delta)$ para o modelo MM*, onde Δ e δ são respectivamente o grau máximo e mínimo de um nodo
	[186] [197]	- diagnosticabilidade de hipercubos e *enhanced hypercubes* - algoritmo de diagnóstico para hipercubos de $O(Nlog_2^2N)$ no pior caso

Tabela A.2: (*Continuação*) Sumário dos resultados do diagnóstico baseado em comparações.

Modelo	Referência	Principais Contribuições
Modelo MM* *(continuação)*	[80]	- diagnosticabilidade de cubos cruzados
	[201]	- algoritmo de diagnóstico $O(Nlog_2^2N)$ para cubos cruzados
	[195]	- diagnosticabilidade de *locally twisted cubes* - algoritmo de diagnóstico $O(Nlog_2^2N)$ para *locally twisted cubes*
	[35, 36]	- diagnosticabilidade de *hypercube-like networks*
	[204]	- diagnosticabilidade de grafos estrela
	[124]	- diagnosticabilidade de *matching composition networks*
	[31]	- (t, k)-diagnóstico para *matching composition networks*
	[29]	- diagnosticabilidade de redes t-conectadas - diagnosticabilidade de redes produto
	[171]	- diagnosticabilidade forte para redes t-regular e t-conectadas
	[104]	- diagnosticabilidade forte para redes produto: hipercubos, *mesh-connected* k-aria n-cubos, *torus-connected* k-aria n-cubos, redes *hyper-Petersen*
	[105]	- diagnosticabilidade forte para *matching composition networks*: cubos cruzados n-dimensionais, *Möbius cubes* e *locally twisted cubes*
	[34]	- diagnóstico baseado em comparações aplicado para redes móveis ad hoc - protocolo Static-DSDP para topologias fixas
	[72]	- protocolo Dynamic-DSDP para redes ad hoc baseado no modelo de Chessa e Santi
	[71, 73]	- diagnóstico baseado em comparações aplicado para redes móveis ad hoc - protocolo Adaptive-DSDP para redes de topologia fixa - protocolo Mobile-DSDP para redes de topologia variantes no tempo
	[37]	- diagnosticabilidade de nodos baseada em estruturas de estrelas estendidas
	[106]	- diagnosticabilidade condicional dos *k-ary n-cubes* é $6n-5$ para $k \geq 4$ e $n \geq 4$
	[102]	- diagnosticabilidade condicional para cubos aumentados n-dimensionais é $6n - 17$ para $n \geq 6$ - diagnosticabilidade forte para os AQ_n é $(2n - 1)$ para $n \geq 5$
	[107]	- diagnosticabilidade forte dos FQ_n é $n + 1$ para $n \geq 5$ - diagnosticabilidade condicional dos FQ_n é $3n - 2$ para $n \geq 5$, 3 para $n = 3$ e 7 para $n = 4$
	[127]	- diagnosticabilidade das 2-MCNs é $t + 2$ - a diagnosticabilidade dos AQ_n é $2n - 1$ para $n \geq 5$ - a diagnosticabilidade dos FQ_n é $n + 1$ para $n \geq 4$
Modelo baseado em Broadcast	[22]	- modelo de comparação completamente distribuído - baseado no modelo MM* para sistemas com broadcast confiável - algoritmos polinomiais para diagnosticar situações estáticas e dinâmicas de falhas

Tabela A.2: (*Continuação*) Sumário dos resultados do diagnóstico baseado em comparações.

Modelo	**Referência**	**Principais Contribuições**
Modelos Distribuídos e Generalizados	[6, 5]	- modelo generalizado e distribuído baseado em comparações: um modelo hierárquico, adaptativo e distribuído baseado no modelo de Sengupta e Dahbura - algoritmo de diagnóstico *Hi-Comp*: requer no máximo $O(N^3)$ comparações e possui latência de $O(log_2 N)$ rodadas no pior caso
	[208]	- modelo generalizado e distribuído baseado em comparações que assume que o resultado das comparações de unidades falhas pode resultar em igualdade - algoritmo de diagnóstico *Hi-Dif* requer no máximo $O(N^2)$ comparações e possui latência de $O(log_2 N)$ rodadas no pior caso
Modelo Probabilístico	[46]	- modelo probabilístico baseado em comparações - considera probabilidades para o resultado de comparações indicar igualdade ou diferença
	[163]	- estratégia baseada na avaliação de múltiplas síndromes
	[89]	- algoritmo $O(log_2 N)$ para a avaliação de múltiplas síndromes
	[128]	- algoritmo provavelmente ótimo para a avaliação de múltiplas síndromes
	[38]	- algoritmo de diagnóstico para sistemas esparsadamente interconectados
Modelo (p, k)-Probabilístico	[156]	- uma tarefa possui k saídas possíveis - cada unidade possui a mesma probabilidade $p < 1/2$ - a probabilidade de se obter igualdade na comparação de uma unidade falha e uma unidade sem-falha ou então de duas unidades falhas é $q = 1/k$ - o problema do diagnóstico e da diagnosticabilidade são NP-difíceis para topologias arbitrária
	[23]	- algoritmo de diagnóstico de tempo polinomial para grafos bipartidos (que incluem hipercubos, grades e florestas) - algoritmo de tempo linear para realizar o diagnóstico ótimo de anéis
Modelos Evolucionários Baseado em Comparações	[68]	- diagnóstico evolucionário baseado em comparações
	[1]	- modelos paralelos de diagnóstico evolucionário
	[74]	- modelos de diagnóstico baseado em comparações com abordagem sobre sistemas imunológicos artificiais
	[75]	- algoritmo neural aplicado a BPNNs (*Backpropagation Neural Networks*)
	[64]	- algoritmo de diagnóstico aplicado a redes neurais *perceptron-based*
	[76]	- algoritmo neural aplicado a BPNNs onde a unidade comparadora pode ser uma das comparadas
	[65]	- algoritmo aplicado a redes neurais *Hopfield*, também analisa o diagnóstico na presença de síndromes parciais
	[77]	- algoritmos neurais aplicados a BPNNs, também realiza o diagnóstico na presença de síndromes parciais
	[66]	- algoritmo de diagnóstico baseado em comparações usando SVMs, também aplicado a síndromes parciais

Tabela A.2: (*Continuação*) Sumário dos resultados do diagnóstico baseado em comparações.

APÊNDICE B

LISTA DE TERMOS, ABREVIAÇÕES E DEFINIÇÕES

Este apêndice mostra – através da Tabela B.1 – uma lista dos termos, abreviações e definições mais importantes, presentes neste trabalho.

Nesta tabela, a primeira coluna (com título *Item*) mostra o termo, abreviação ou definição, em uma forma ordenada. A coluna *Descrição* apresenta uma descrição resumida de cada item. Por fim, a terceira coluna lista o número das principais páginas que contêm a definição ou detalhes sobre o item correspondente.

Item	**Descrição**	**Páginas**
(i, j)	Uma aresta de E que conecta as unidades i e j.	26
$(j, k)_i$	Um teste realizado pela unidade i através do envio de uma tarefa às unidades j e k e da comparação da saída retornada.	26, 51
$r((j, k)_i)$	Resultado da comparação das saídas das unidades j e k pela unidade i.	27, 51
(n, k)-hipercubo	*Enhanced hypercubes* construído através da adição de 2^{n-1} *links* extras ao n-hipercubo (H_n) correspondente.	150
δ	Grau da unidade de menor grau no sistema.	35, 36
Δ	Grau da unidade de maior grau no sistema.	36
$\Gamma(i)$	$\Gamma(i) = \{j \mid i$ e j são comparadas$\}$.	25
λ	Número de anéis configurados no Fireflies.	93
$\kappa(G)$	$\kappa(G) = min\{\|V'\|$ tal que $V' \subseteq V$ e $G - V'$ não é conectado$\}$.	167, 169, 173
$\mathcal{F}(v)$	Em um algoritmo genético de diagnóstico, denota o conjunto de nodos falhos de acordo com o cromossomo v.	188
σ	Síndrome do sistema, ou síndrome de comparações.	26
$\sigma(F)$	Conjunto de síndromes que podem ser geradas se F é o conjunto de nodos falhos.	32
$\xi(G)$	Conjunto de componentes maximais do grafo G.	53
2-MCN	Uma *two-matching composition network*, ou $G(G_1, G_2; PM_2)$.	166
$a_{i,j}$	Resultado do teste da unidade i sobre a unidade j, no modelo PMC.	11, 13
AFS	*Allowable Fault Set*, ou possível conjunto de unidades falhas.	33, 52
AIS	*Artificial Immune System*, ou sistema imunológico artificial.	188
algoritmo *Adaptive-DSD*	Algoritmo de diagnóstico adaptativo e distribuído em nível de sistema.	15
algoritmo completo	Definição de um algoritmo que consegue identificar todas as unidades falhas do sistema.	14

Tabela B.1: Lista de termos, abreviações e definições.

Item	Descrição	Páginas
algoritmo correto	Definição usada quando o estado das unidades diagnosticadas pelo algoritmo é identificado corretamente.	14
algoritmo *Diag*	Novo algoritmo $O(t^2\Delta N)$ proposto para o diagnóstico de falhas em sistemas de topologia arbitrária com base no modelo MM*.	59
algoritmo *DIAGNOSIS*	Algoritmo de diagnóstico baseado em comparações apresentado por Sengupta e Dahbura para sistemas de topologia arbitrária com base no modelo MM*.	37
algoritmo *Diagnostico*	Algoritmo de diagnóstico executado pelo *tracker*.	101
algoritmo *Dynamic*	Algoritmo apresentado por Blough e Brown para diagnosticar sistemas sobre situações dinâmicas de falhas.	181
algoritmo *EVENT-SELF*	Algoritmo de diagnóstico distribuído em nível de sistema; extensão do algoritmo *NEW-SELF*.	15
algoritmo *Genetic-Comparison-Diagnosis*	Algoritmo genético de diagnóstico baseado em comparações proposto por Elhadef e Ayeb.	187
algoritmo *Hi-ADSD*	Algoritmo hierárquico, adaptativo e distribuído em nível de sistema; constrói *clusters* de tamanhos progressivos.	16
algoritmo *Hi-ADSD with Timestamps*	Algoritmo hierárquico, adaptativo e distribuído em nível de sistema; emprega *clusters* com $N/2$ nodos.	16
algoritmo *Hi-Comp*	Algoritmo de diagnóstico hierárquico adaptativo e distribuído em nível de sistema baseado em comparações.	46
algoritmo *Hi-Dif*	Algoritmo de diagnóstico hierárquico adaptativo e distribuído em nível de sistema baseado em comparações.	49
algoritmo *MM*_DIAG*	Algoritmo de diagnóstico baseado em comparações apresentado por Yang e Tang para sistemas de topologia arbitrária com base no modelo MM*.	42
algoritmo *ModuloComparador*	Algoritmo implementado pelo módulo comparador, que é executado em todos os *peers* da rede.	99, 112
algoritmo *NEW-SELF*	Algoritmo de diagnóstico distribuído em nível de sistema; extensão do algoritmo *SELF*.	15
algoritmo *Peer*	Parte de código adicionada ao algoritmo dos *peers*.	112
algoritmo *RDZ*	Algoritmo distribuído para sistemas de topologias arbitrárias.	17
algoritmo *SELF*	Algoritmo de diagnóstico distribuído em nível de sistema.	14
algoritmo *Static-Complete*	Algoritmo apresentado por Blough e Brown para o diagnóstico do sistema sobre a situação de falhas estáticas, para o modelo de diagnóstico baseado em comparações com *broadcast*.	181
algoritmo *Static-Partial*	Algoritmo apresentado por Blough e Brown para o diagnóstico de situações onde somente uma síndrome parcial está disponível.	180
ANN	*Artificial Neural Network*, ou rede neural artificial.	189
AQ_n	Um *n-dimensional augmented cube*, ou cubo aumentado n-dimensional.	167
árvore	Uma das topologias empregadas em redes P2P.	86
BC *Network*	Uma rede BC (*Bijective Connection Network*).	175
$BF(k, r)$	Uma rede borboleta k-aria r-dimensional.	153
BPNN	*Backpropagation Neural Network*.	190
C	Conjunto de todas as comparações $(j, k)_i$ realizadas no sistema.	23, 26, 51
caminho $P[v_0, v_z]$	Um caminho em G onde $\{v_0, v_i, \ldots, v_z\} \subseteq V$, é uma sequência de vértices distintos tal que qualquer par de vértices consecutivos são adjacentes e v_0 e v_z são os vértices finais do caminho.	53
chunk	Parte (ou pedaço) do conteúdo que é transmitido na rede P2P.	85
ciclo de testes	Uma aplicação do número máximo de comparações no sistema.	30
cid	Identificador de um *chunk* , ou *chunk identifier*.	95

Tabela B.1: (*Continuação*) Lista de termos, abreviações e definições.

Item	Descrição	Páginas
classes de falhas	Classificação das razões pelas quais uma unidade pode se tornar falha.	21
clusters	Grupos virtuais de nodos.	16
$CompF_{i,j}$	Conjunto com três unidades $\{i, j, k\}$ tal que $\exists r((j,k)_i) = 1$ e $k \in FF_i$.	54
$CompFF_{i,j}$	Conjunto com três unidades $\{i, j, k\}$ tal que $\exists r((j,k)_i) = 0$.	54
componente conexo máximo de G	Um subgrafo $G_x = (V_x, E_x)$ onde $V_x \subseteq V$, $E_x = \{(j,k) \in E \mid j,k \in V_x\}$ tal que qualquer par de vértices $v_a, v_b \in V_x$ são conectados um ao outro por pelo menos um caminho $P[v_a, v_b]$ e não existe nenhum par de vértices v_x, v_y tal que $v_x \in V_x$, $v_y \in V - V_x$ e existe a aresta $(v_x, v_y) \in E$.	53
CQ_n	Um cubo cruzado n-dimensional.	155
cromossomo	Em um algoritmo genético de diagnóstico, é a representação de potenciais soluções para o problema do diagnóstico.	185
$d(i)$	$d(i) = \|N(i)\|$ é o grau (ou ordem) da unidade i.	26, 51
$d_{i,j}$	Distância de diagnóstico entre o nodo i e o nodo j.	46
diagnóstico com reparação	Mesmo que diagnóstico sequencial: pelo menos uma unidade pode ser identificada e ser reparada ou substituída, e assim os testes podem continuar.	13
diagnóstico em um passo	Toda unidade falha do sistema pode ser identificada desde que o número de unidades falhas não seja maior que t.	12
diagnóstico sequencial	Pelo menos uma unidade pode ser identificada e ser reparada ou substituída, e assim os testes podem continuar.	12
distância de diagnóstico	Menor distância entre dois nodos no grafo $T(S)$.	46
E	Conjunto de arestas do grafo $G = (V, E)$. Cada aresta representa o enlace de comunicação entre um par de unidades.	18, 26, 51
$E(G_x)$	O conjunto de arestas do grafo G_x.	159
$\overline{E_Z}$	Considerando $\overline{G_Z} = (V - Z, \overline{E_Z})$, $\overline{E_Z} = \{(j,k) \in E \mid j,k \in V - Z\}$.	53
$ES(x; n)$	Uma estrela estendida de ordem n no nodo x.	161
evento	Uma mudança de estado de um nodo.	46
F	Conjunto de todas as unidades falhas.	26, 51
F_i	Conjunto das unidades falhas pela visão da unidade i; se $r((j,k)_i) = 1$ e $k \in FF_i$ então $j \in F_i$.	54
F'_i	Conjunto definido como segue: $\forall u \in FF'_i, F'_i \leftarrow F'_i \cup F_u$.	55
F'_i máximo	F'_i é máximo se i é sem-falha e $\forall j \in V$, $j \neq i$, $\|F'_j\| \leq \|F'_i\|$.	56
$F_i^\diamond$	Conjunto definido como segue: se $i \in F_v$ então $v \in F_i^\diamond$.	54
FF_i	Conjunto das unidades sem-falha pela visão da unidade i; se $r((j,k)_i) = 0$ então $j,k \in FF_i$.	54
FF'_i	Conjunto definido como segue: i está sempre em FF'_i; $j \in FF'_i$ se existe pelo menos um caminho $P[i,j]$ da unidade i para a unidade j tal que para todo par de vértices distintos e consecutivos (v_1, v_2) em $P[i,j]$, $v_2 \in FF_{v_1}$.	55
$FF_i^\diamond$	Conjunto definido como segue: se $i \in FF_v$ então $v \in FF_i^\diamond$.	54
Fireflies	Protocolo escalável que cria uma rede *overlay* tolerante a intrusões.	92
FQ_n	Um *folded n-hypercube*, ou hipercubo dobrado n-dimensional.	167
$FT(v)$	A função de aptidão de um cromossomo v, em um algoritmo genético de diagnóstico.	186
fonte, ou servidor fonte	Entidade responsável por gerar e disseminar o conteúdo que é transmitido em uma rede P2P.	85
função is_AFS	Função utilizada pelo algoritmo $Diag$ e que verifica se um determinado conjunto de unidades é um AFS.	57

Tabela B.1: (*Continuação*) Lista de termos, abreviações e definições.

Item	Descrição	Páginas
G, ou $G = (V, E)$	Grafo que representa o sistema S, V é o conjunto de vértices e E o conjunto de arestas.	18, 26, 51
$G(G_1, G_2; PM_2)$	Uma *two-matching composition network*, ou 2-MCN.	166
$G(\tau) = (V, L(\tau))$	O grafo que representa o sistema no tempo τ em redes ad hoc.	193
$\overline{G_Z}$, ou $\overline{G_Z} = (V - Z, \overline{E_Z})$	Subgrafo resultante da remoção de um conjunto de vértices Z de V.	53
$G[V']$, ou $G[V_i]$	Um subgrafo de $G = (V, E)$ *induzido por* V' (ou V_i).	53, 175
GA	*Genetic Algorithm*, ou algoritmo genético.	185
grau (ou ordem)	Número de arestas adjacentes a um determinado nodo.	22, 26, 51
$H(i, j)$	Distância de Hamming entre os nodos i e j em um hipercubo H_n.	150
H_n	Hipercubo n-dimensional, ou n-hipercubo.	150
HL_n	Uma rede *hypercube-like* n-dimensional.	159
HP_n	Uma rede *hyper-Petersen* n-dimensional.	172
informações de diagnóstico	Informações recebidas pelo nodo testador a partir do nodo testado; incluem o estado de outros nodos do sistema.	16
intervalo de testes	Intervalo no qual cada nodo executa pelo menos um teste.	15
janela de disponibilidade	Lista que indica quais *chunks* cada *peer* possui disponíveis para envio a seus vizinhos.	93
janela de interesse	Lista que indica quais *chunks* cada *peer* ainda precisa receber.	93
$\mathbb{L}_n$	Uma família de redes BC n-dimensionais, ou família de redes X_n.	175
L, ou $L(V_1, V_2)$	Uma correspondência perfeita (*perfect matching*) entre os vértices de dois grafos.	164
$L(\tau)$	O conjunto dos enlaces lógicos existentes no tempo τ em redes ad hoc.	193
latência de diagnóstico	Número de rodadas de testes necessárias para que todos os nodos sem-falha completem o diagnóstico do sistema.	16, 30
LTQ_n	Um *locally twisted cube* n-dimensional.	158
m	Nos modelos probabilísticos de diagnóstico baseado em comparações, é o número total de diferentes possíveis saídas incorretas que um processador falho pode produzir para uma tarefa.	183
M, ou $M = (V, C)$	Multigrafo que representa as comparações realizadas no sistema.	26, 51
M_k^n	Um *mesh-connected* k*-ary* n*-cube*.	171
$M(S)$	Multigrafo que representa os testes executados nos modelos generalizados de diagnóstico distribuído baseado em comparações.	46
MANETs	*Mobile Ad hoc NETworks*, ou redes móveis ad hoc.	192
$MASF(\sigma)$	AFS mínimo de σ.	33
MCN_i	uma *matching composition network* i-dimensional.	164
mesh	Uma das topologias empregadas em redes P2P.	86
modelo BGM	Modelo de diagnóstico em nível de sistema proposto por Barsi, Grandoni e Maestrini.	12
modelo BSC	Modelo *Bounded Symmetric Comparison*.	24
modelo MM	Modelo de diagnóstico baseado em comparações apresentado por Maeng e Malek.	26, 51
modelo MM*	Caso especial do modelo MM onde cada unidade compara todo par de unidades vizinhas.	30, 52
modelo PMC	Modelo de diagnóstico em nível de sistema apresentado por Preparata, Metze e Chien.	26
modelos KH1 e KH2	Dois modelos de diagnóstico baseado em comparações apresentados por Kreutzer e Hakimi.	24

Tabela B.1: (*Continuação*) Lista de termos, abreviações e definições.

Item	Descrição	Páginas
N	Número de unidades no sistema S.	10, 18, 26, 51
$N(i)$	Conjunto de unidades vizinhas de i.	26, 51
$N(i, \tau)$	Conjunto de vizinhos de i no tempo τ em uma rede ad hoc, também pode ser denotado simplesmente por $N(i)$.	193
observador central	Entidade externa que realiza o diagnóstico do sistema.	11, 18, 26
ordem (ou grau)	Número de arestas adjacentes a um determinado nodo.	22, 26, 51
p	Nos modelos probabilísticos de diagnóstico baseado em comparações, é a probabilidade de que uma unidade falha produza a saída correta para uma tarefa.	183
$P(W_i)$	Nos modelos probabilísticos de diagnóstico baseado em comparações, é a probabilidade de que uma unidade falha produza a saída incorreta W_i para uma tarefa.	183
P_i	Conjunto das unidades pendentes pela visão da unidade i; se $\nexists r((j,k)_i) = 0$ então $P_i = N(i)$, caso contrário $P_i = \emptyset$.	55
PM_2	Um conjunto composto com duas correspondências perfeitas diferentes.	166
peer	Um usuário da rede P2P.	85
protocolo *Adaptive-DSDP*	*Adaptive Distributed Self-Diagnosis Protocol*; protocolo apresentado para o modelo de diagnóstico de Elhadef, Boukerche e Elkadiki, para redes de topologia fixa.	197
protocolo *Mobile-DSDP*	*Mobile Distributed Self-Diagnosis Protocol*; protocolo apresentado para o modelo de diagnóstico de Elhadef, Boukerche e Elkadiki, para redes de topologia variante no tempo.	197
protocolo *Static-DSDP*	*Static Distributed Self-Diagnosis Protocol*; protocolo apresentado para o modelo de diagnóstico em redes ad hoc de Chessa e Santi.	194
pull-based	Uma das estratégias de transmissão de dados em redes P2P; um dado é enviado por um *peer* a outro apenas se ocorrer uma requisição.	87
push-based	Uma das estratégias de transmissão de dados em redes P2P; os dados são transmitidos de um *peer* para outro sem que ele seja solicitado.	87
push-pull-based	Uma das estratégias de transmissão de dados em redes P2P; combina ambas as estratégias *push-based* e *pull-based*.	88
$R_{u,i}$, ou R_i^u	Notação que representa o resultado esperado de uma tarefa i gerada pela unidade u nos modelos de diagnóstico baseados em comparações aplicados a redes ad hoc.	194, 198
redes produto	Uma rede gerada pela aplicação da operação de produto cartesiano de grafos a redes de fator.	169
rodada de testes	Período de tempo no qual todos os nodos do sistema executam todos os seus testes pelo menos uma vez.	15, 46
S	Sistema assumido pelo diagnóstico em nível de sistema.	10, 18, 26, 51
$\mathcal{S}$	Conjunto de unidades suspeitas; conjunto que consiste das três unidades $\{s_1, s_2, s_3\}$ envolvidas em uma comparação $(s_2, s_3)_{s_1} \in C$, tal que uma das duas condições de verificação do AFS não são satisfeitas.	56
S_n	Um grafo estrela n-dimensional, ou *n-star*.	162
$S_{t,N}$	Grafo mínimo para diagnosticar até t unidades falhas em um sistema de N unidades.	29
servidor fonte	Entidade responsável por gerar e disseminar o conteúdo que é transmitido em uma rede P2P.	85

Tabela B.1: (*Continuação*) Lista de termos, abreviações e definições.

Item	Descrição	Páginas
sistema t-diagnosticável	Sistema no qual todas as unidades falhas podem ser identificadas desde que o número de unidades falhas seja menor ou igual a t.	32, 52
síndrome	Conjunto com o resultado de todos os testes realizados no sistema.	11, 26, 51
síndrome de comparações	Conjunto com o resultado de todos os testes (ou comparações), nos modelos de diagnóstico baseados em comparações.	26, 51
síndrome de testes	Conjunto com o resultado de todos os testes, nos modelos de diagnóstico em nível de sistema baseados no modelo PMC.	11
SVMs lineares	*Linear support vector machines*.	191
t	Número máximo de unidades falhas permitido no sistema.	12, 28
t-AFS	AFS com no máximo t unidades.	33, 52
$t(G)$	Valor t tal que o sistema é t-diagnosticável.	174
$t_c(G)$	Valor t tal que o sistema é condicionalmente t-diagnosticável.	174
$t_s(G)$	Valor t tal que o sistema é fortemente t-diagnosticável.	170
T_k^n	Um *torus-connected k-ary n-cube*.	172
T_i	Notação usada para representa uma tarefa i nos modelos de diagnóstico baseados em comparações aplicados a redes ad hoc.	194
$T_i(S)$	Grafo direcionado baseado na $T(S)$ que mostra como o nodo i obtêm informações de diagnóstico.	46
T_{cid}	Conjunto mantido pelo *tracker*, e que possui o mesmo formato do conjunto $U_{i,cid}$.	97
$T(S)$	Grafo que representa estratégia de testes nos algoritmos *Hi-Comp* e *Hi-Dif* dos modelos generalizados de diagnóstico baseado em comparações.	46, 49
tracker	Entidade central confiável, que nunca falha, e é acessível por todos os *peers* da rede P2P.	95
u_i	Notação que representa a unidade u_i, que também pode ser referenciada por *unidade i*, *nodo i*, ou mesmo *processador i*.	10
$U_{i,cid}$	Conjunto que contém o conteúdo de cada diferente *chunk* recebido pelo módulo comparador e também o identificador dos *peers* que retornaram o *chunk* com aquele exato conteúdo. Possui o formato $U_{i,cid} = \{(chunk_a, \{peer_i, peer_j, ...\}), (chunk_b, \{peer_k, ...\}), ...\}$.	95, 96, 109
V	Conjunto de vértices do grafo $G = (V, E)$. Cada vértice corresponde a um processador, um nodo, ou uma unidade do sistema.	18, 26, 51
$V(G_x)$	O conjunto de vértices do grafo G_x.	159
W_i	Nos modelos probabilísticos de diagnóstico baseado em comparações, é uma das m possíveis saídas incorretas para uma tarefa, $1 \leq i \leq m$.	183
X_n	Uma rede BC n-dimensional (*n-dimensional BC Network*).	175

Tabela B.1: (*Continuação*) Lista de termos, abreviações e definições.