GIORGIA DE OLIVEIRA MATTOS

DIAGNÓSTICO DE REDES DE TOPOLOGIA ARBITRÁRIA: UM ALGORITMO BASEADO EM INUNDAÇÃO DE MENSAGENS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, Curso de Mestrado em Informática, Departamento de informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA
2001

UFPR
Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática da aluna ***Giorgia de Oliveira Mattos***, avaliamos o trabalho intitulado "***Um Algoritmo Baseado em Inundação para Diagnóstico de Redes de Topologia Arbitrária***", cuja defesa foi realizada no dia 16 de março de 2001. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 16 de março de 2001.

Prof. Dr. Elias Procópio Duarte Júnior
Presidente - Orientador

Prof. Dr. Carlos Alberto Maziero
Membro Externo - PUC/PR

Prof. Dr. Roberto André Hexsel
DINF/UFPR

Agradecimentos

À Deus pela existência humana.

Ao meu pai e à minha mãe, que vivendo em família, sempre tiveram como meta a educação e a formação de suas filhas.

Às minhas irmãs, Daphine e Demetra pelo apoio e compreensão na fase final desse trabalho.

Ao João Israel Bernardo pelo carinho e apoio constantes.

Ao professor Elias Procópio Duarte Jr., meu orientador, pela sabedoria e dedicação.

Aos demais professores do curso de mestrado em Informática da Universidade Federal do Paraná pelos ensinamentos.

À mestre Andréa Weber, funcionária e pesquisadora do Departamento de Informática da UFPR e ao José Marcelo A. P. Cestari, amigo do curso de mestrado, pelas simulações realizadas e compartilhamento dos resultados obtidos, contribuindo para este trabalho.

À todos os amigos do curso de mestrado, professores do Curso de Tecnologia em Informática do CEFET-PR/Unidade de Pato Branco, pelo apoio e incentivo.

À Universidade Federal do Paraná – UFPR, pela oportunidade de aprender.

Ao Centro Federal de Educação Tecnológica do Paraná – Unidade de Pato Branco, através do convênio com a UFPR, pelos esforços dedicados à capacitação docente.

"A razão cardeal de toda a superioridade humana é sem dúvida a vontade. O poder nasce do querer. Sempre que uma pessoa aplica a veemência e a perseverante energia de sua alma a um fim, ela vencerá os obstáculos, e se não atingir o alvo, fará pelo menos coisas admiráveis."

(José de Alencar)

Conteúdo

Resumo

Em uma rede de computadores tanto nodos como enlaces podem falhar. Este trabalho apresenta um algoritmo de diagnóstico distribuído de redes de topologia arbitrária que permite a monitoração da rede. Os nodos testam os enlaces que os conectam a outros nodos. Quando um nodo detecta uma falha este dissemina em paralelo, para os seus vizinhos, uma mensagem de disseminação contendo informações sobre a falha. Os vizinhos, ao receberem a mensagem, comparam as suas informações locais de diagnóstico com a informação contida na mensagem. Se a informação já é conhecida, a mensagem redundante é descartada, caso contrário as informações locais são atualizadas e disseminadas. As mensagens redundantes empregadas são, na verdade, consideradas uma vantagem do algoritmo, quando comparado a outras abordagens. Os algoritmos de diagnóstico são justamente usados para permitir que os nodos sem falhas possam determinar a situação do sistema quando o sistema está parcialmente inoperante. Desta forma é fundamental que tais algoritmos sejam tolerantes a falhas. Se um nodo recebe uma mensagem redundante, é porque existem dois caminhos disjuntos entre o nodo que gerou a mensagem e o nodo que a recebe. Assim, se durante a disseminação da mensagem de diagnóstico novos eventos de falha ocorrerem na rede, a redundância vai permitir que o algoritmo tolere falhas de caminhos, tantas quantas são as mensagens redundantes. Por outro lado, as mensagens são pequenas, e o número máximo de mensagens por evento é 2*L, onde L é o número de enlaces no sistema. O algoritmo não trabalha com eventos dinâmicos e a falha de um enlace não particiona a rede. Simulações são realizadas em diversas topologias dentre elas a topologia $D_{1,2}$, hipercubo, grafos randômicos e a topologia da RNP. Os resultados mostram que a latência do algoritmo é proporcional ao diâmetro da rede. Comparações com outros algoritmos são apresentadas. Os parâmetros analisados são o total de mensagens de disseminação, o número de mensagens redundantes e o tempo necessário para realizar o diagnóstico.

Abstract

In this work a system-level distributed diagnosis algorithm for general topology networks is presented. The algorithm has three phases: test, dissemination and diagnosis itself. In each testing interval, nodes execute tests on their neighbors. After an event is detected, event information is disseminated in parallel to the rest of the network. Dissemination is based on message flooding. When a node receives a message, it checks whether the diagnostic information contained in the message is new or already known. If the information is new, the node forwards the message to its neighbors, and updates its local information. We assume that faults do not partition the network. Experimental results obtained through simulation algorithm in several topologies are presented, for instance the $D_{1,2}$ graph with 9 nodes, hypercubes of 16, 64, and 128 nodes, random graphs and the RNP topology. Results show that the algorithm's latency is proportional to the diameter of the network. Comparison with other algorithms show that the algorithm always produces the best latency. Although the number of redundant messages generated is large, this is considered an advantage of the algorithm, as it works even in the presence of new faults in the network.

Capítulo 1

Introdução

O uso das redes de computadores vem aumentando nas mais diversas áreas de aplicação. As redes são cada vez maiores e mais complexas, sendo compostas por equipamentos heterogêneos de diversos tipos. Desta forma, surgiu a necessidade de ferramentas automatizadas que permitam a monitoração eficiente de uma rede.

A maioria dos sistemas de monitoração de redes de computadores são *centralizados* ou *hierárquicos* [6]. Estes sistemas não são tolerantes a falhas, pois, se um monitor falha, a rede deixa de ser monitorada. Além disso, a concentração de mensagens de gerência em um único nodo pode acarretar efeitos negativos no seu desempenho. Neste trabalho apresentamos uma estratégia que pode ser usada para implementar a monitoração distribuída de falhas de redes WAN. Esta abordagem é baseada em *diagnóstico em nível de sistema.*

Este capítulo apresenta o diagnóstico em nível de sistema, o diagnóstico em redes de topologia arbitrária e uma visão geral do novo algoritmo.

1.1 Diagnóstico em Nível de Sistema

Há mais de 30 anos, o problema de detecção de falhas em sistemas computacionais tem sido o foco da pesquisa em *diagnóstico em nível de sistema* [17]. O

objetivo dos algoritmos de diagnóstico em nível de sistema é permitir que todos os nodos sem-falha determinem o estado de todos os nodos do sistema. Dois *estados* são possíveis, o estado de *falha* e o estado de *sem-falha*. Cada nodo executa testes em outros nodos, e os nodos sem-falha são capazes de determinar corretamente o estado de todos os nodos testados por eles.

Os algoritmos para diagnóstico em nível de sistema podem ser divididos em duas categorias: os que assumem que entre todo par de nodos na rede existe um único canal de comunicação, ou seja, o grafo do sistema é completo; e os que permitem que a rede seja de topologia arbitrária.

O modelo original de diagnóstico foi apresentado por Preparata, Metze e Chien [18] e também por Hakimi e Amin [11]. Em um sistema com N nodos, cada nodo testa um subconjunto de seus vizinhos, onde um nodo pode estar sem-falha ou falho. Todos os resultados dos testes são enviados a um observador central que é responsável por interpretá-los e completar o diagnóstico do sistema.

Posteriormente, os algoritmos de diagnóstico em nível de sistema passaram a empregar testes adaptativos, onde a escolha dos próximos testes depende dos resultados obtidos de testes anteriores, e não dependem de um padrão fixo. Hakimi e Nakajima chamam este método de diagnóstico *adaptativo* [12].

No início da década de 80, Kuhl e Reddy [14, 15] apresentaram o diagnóstico em nível de sistema *distribuído*. Um algoritmo distribuído de diagnóstico permite aos nodos sem-falha do sistema determinar quais nodos estão falhos e quais estão sem falha. O primeiro algoritmo de diagnóstico em nível de sistema ao mesmo tempo adaptativo e distribuído é o algoritmo *Adaptive–DSD* [2, 5]. Mais tarde, o algoritmo *Hierarchical Adaptive Distributed System-Level Diagnosis* (Hi-ADSD) [9] foi apresentado e implementado com SNMP *(Simple Network Management Protocol)*. Os algoritmos

Adaptative-DSD e Hi-ADSD pressupõem que o sistema é representável por um grafo completo.

1.2 Diagnóstico de Redes de Topologia Arbitrária

Os algoritmos que realizam diagnóstico em nível de sistema de redes de topologia arbitrária, assumem que entre dois nodos do sistema pode haver ou não um canal de comunicação direto. A execução de um algoritmo de diagnóstico de redes de topologia arbitrária está dividida em três etapas: *teste*, *disseminação* e *diagnóstico*. Na etapa de teste todos os nodos do sistema testam os enlaces que os conectam a outros nodos. Na etapa de disseminação ocorre o envio de mensagens para todos os nodos da rede, informando sobre a mudança de estado ocorrida. Na etapa de diagnóstico todos os nodos sem-falha identificam o estado de todos os nodos da rede. Desta forma, usando as informações armazenadas sobre o estado dos enlaces, um nodo sem-falha executa um algoritmo para determinar a conectividade do grafo que representa a rede.

Os algoritmos detectam falhas nos canais de comunicação. Para estes é possível determinar se um canal de comunicação está falho ou sem-falha, mas é impossível distinguir se um nodo está falho ou se todos os canais de comunicação que levam ao nodo é que estão falhos.

Bagchi e Hakimi [1] apresentam um algoritmo para diagnóstico de redes de topologia arbitrária onde o número de mensagens trocadas entre os nodos é considerado ótimo. Porém, o algoritmo não é executado on-line, não permitindo o monitoramento dinâmico e contínuo da rede, ou seja, durante a sua execução, um nodo não pode falhar ou recuperar-se.

Bianchini e outros [3, 4] apresentam o algoritmo *Adapt* para o diagnóstico distribuído de redes de topologia arbitrária. A rede é representada por uma árvore onde

cada nodo da árvore é testado por seu pai, exceto o nodo raiz, que é testado por um de seus filhos. O algoritmo *Adapt* é executado on-line. Quando ocorre um evento, os nodos sem-falha se organizam de forma a manter o grafo conexo. A informação contendo o evento ocorrido, é propagada pela rede usando uma árvore de busca em profundidade que é naturalmente seqüencial. O caminho que a mensagem percorre na rede determina a nova topologia de rede baseada em árvore.

Rangarajan e outros [19] apresentam um novo algoritmo para diagnosticar falhas de redes de topologia arbitrária. Este algoritmo é também chamado *RDZ*, das iniciais dos autores. O algoritmo RDZ é executado on-line, garante o menor número de testes por nodo e apresenta a menor latência de diagnóstico em função do uso de uma estratégia de propagação de mensagens em paralelo. Porém, o RDZ não identifica falhas em certas configurações, impossibilitando o seu uso em gerência de falhas de redes.

Em 1997, Duarte e outros [6, 10] apresentam o algoritmo NBND *(Non-Broadcast Network Diagnosis)* para redes ponto-a-ponto de topologia arbitrária que resolve o problema de detecção de falhas em configurações que o algoritmo RDZ não detecta. Este algoritmo permite o diagnóstico de *time-outs* de canais de comunicação, calculando a conectividade da rede usando o menor número de testes possível, um por canal, sendo a latência proporcional ao diâmetro da rede. O algoritmo NBND foi implementado para monitoração baseada em SNMP [21].

Duarte e Cestari [7] apresentam o algoritmo do Agente Chinês, que é um algoritmo distribuído para detecção de falhas de redes de topologia arbitrária. Um agente móvel permite o diagnóstico do sistema percorrendo todos os nodos e todos os canais de comunicação, seguindo o caminho do carteiro chinês. As etapas de teste e disseminação são seqüenciais, o que pode causar impacto na latência do algoritmo quando comparado a

outros algoritmos, como por exemplo o algoritmo NBND e o algoritmo proposto neste trabalho.

1.3 O Novo Algoritmo

Neste trabalho é apresentado um novo algoritmo de diagnóstico distribuído de redes de topologia arbitrária que detecta falhas nos canais de comunicação. Em um intervalo de testes todos os nodos testam o enlace para os outros nodos. Um nodo da rede, ao descobrir um evento de falha, envia paralelamente aos nodos alcançáveis dessa rede uma mensagem de disseminação avisando-os do novo evento ocorrido. O algoritmo apresenta um mecanismo para descartar mensagens redundantes ou seja, um nodo ao receber uma mensagem de disseminação compara as informações recebidas com as suas informações locais de diagnóstico, se a informação já for conhecida pelo nodo, este descarta a mensagem pois ela é redundante. O algoritmo não permite a ocorrência de eventos dinâmicos, ou seja durante a execução do algoritmo um nodo não pode falhar ou recuperar-se antes que o evento anterior tenha sido disseminado pela rede e a falha de um enlace não particiona a rede.

As mensagens redundantes empregadas são, na verdade, consideradas uma vantagem do algoritmo, quando comparado a outras abordagens. Os algoritmos de diagnóstico são justamente usados para permitir que os nodos sem falhas possam determinar a situação do sistema quando o sistema está parcialmente inoperante. Desta forma é fundamental que tais algoritmos sejam tolerantes a falhas.

Se um nodo recebe uma mensagem redundante, é porque existem dois caminhos disjuntos entre o nodo que gerou a mensagem e o nodo que a recebe. Desta forma, se durante a disseminação da mensagem de diagnóstico novos eventos de falha ocorrerem na rede, a redundância vai permitir que o algoritmo tolere falhas de caminhos,

tantas quantas são as mensagens redundantes. Por outro lado, as mensagens são pequenas, e o número máximo de mensagens por evento é 2*L, onde L é o número de enlaces no sistema.

Um exemplo da execução do algoritmo é apresentado na figura 1. Os testes são realizados em intervalos de testes, assim em um intervalo de testes os nodos testam os enlaces aos quais estão conectados. Desta forma, o nodo 1 testa o enlace para o nodo 2 e nodo 4. O nodo 2 testa o enlace para o nodo 1 e nodo 3. O nodo 3 testa o enlace para o nodo 2 e nodo 4. Finalmente o nodo 4 testa o enlace para o nodo 1 e nodo 3. Se nenhuma mudança de estado foi detectada os testes são reiniciados. No próximo intervalo de testes o nodo 2 ao testar o enlace para o nodo 3 detecta uma falha. Neste momento o nodo 2 prepara-se para iniciar a disseminação da mensagem contendo informações sobre a falha para os seus vizinhos, no próximo intervalo de tempo. O nodo 2 dissemina a mensagem para os seus vizinhos, os quais por sua vez também disseminam a mensagem para os seus vizinhos, exceto para aquele de onde veio a mensagem. O nodo 3 ao tentar disseminar a mensagem para o nodo 2 também detecta a falha e inicia o processo de disseminação da mensagem, mensagem esta diferente daquela enviada pelo nodo 2. O algoritmo termina quando todos os nodos processam e disseminam as mensagens.

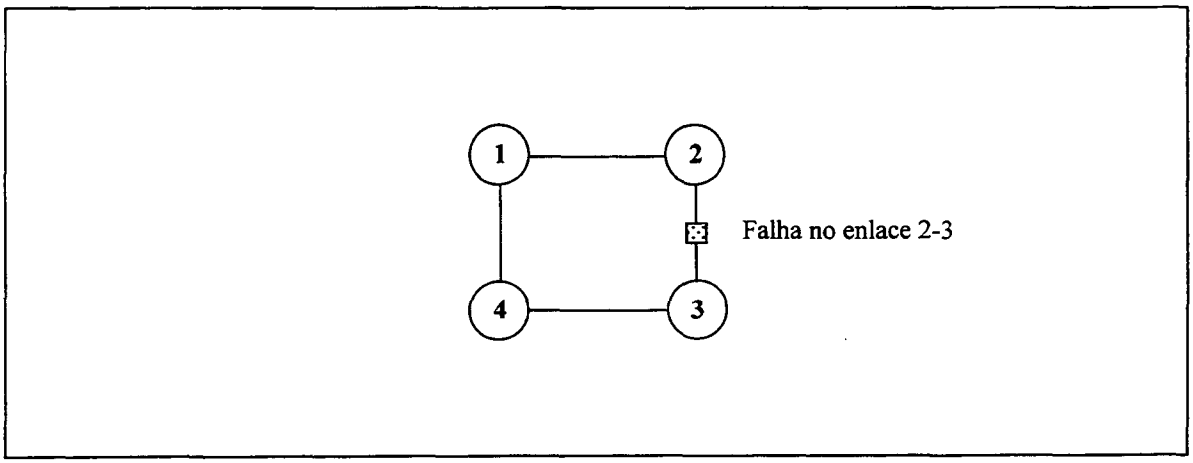


Figura 1: Grafo exemplo da execução do algoritmo.

São apresentados resultados experimentais obtidos através de simulações em redes de diversas topologias. Em cada experimento falhas de um enlace foram simuladas, assim os dois nodos conectados por um enlace detectam o evento em instantes de tempo distintos e disseminam mensagens distintas.

O primeiro experimento realizado é em uma rede exemplo, representada por um grafo de 7 vértices. O segundo experimento é em uma rede de topologia $D_{1,2}$ de 9 vértices. O terceiro experimento realizado são em redes de topologia hipercubo de 16, 64 e 128 vértices. O quarto experimento é em uma rede gerada randomicamente com 50 vértices. Resultados de simulação em 50 redes randômicas também são apresentados. O quinto experimento realizado é a topologia da RNP (Rede Nacional de Pesquisa).

Os resultados obtidos nas simulações mostram que o número de mensagens redundantes geradas é, na média, menor que o máximo possível, isto é, o dobro do número de enlaces. A latência do algoritmo é proporcional ao diâmetro da rede, e este emprega o menor número de testes, um por canal de comunicação por intervalo de testes. Comparações com outros algoritmos também são apresentadas. Os parâmetros utilizados nas comparações são o total de mensagens de diagnóstico, o número de mensagens redundantes e o tempo para realizar o diagnóstico.

O restante deste trabalho está estruturado da seguinte forma: o capítulo 2 apresenta os algoritmos de diagnóstico de redes de topologia arbitrária; no capítulo 3 o algoritmo baseado em inundação de mensagens é descrito; o capítulo 4 apresenta resultados experimentais obtidos através de simulações, bem como as comparações com outros algoritmos; o capítulo 5 conclui o trabalho. O apêndice contém o código do programa usado nas simulações.

Capítulo 2

Diagnóstico de Redes de Topologia Arbitrária

Existem duas categorias de algoritmos de diagnóstico. Uma delas realiza diagnóstico de redes nas quais existe um canal de comunicação direto entre quaisquer dois nodos da rede. A outra realiza diagnóstico de redes de topologia arbitrária; entre dois nodos do sistema pode haver ou não um canal de comunicação direto. Neste caso, para que dois nodos se comuniquem eles devem usar nodos intermediários. Os termos rede e sistema são usados como sinônimos neste trabalho, bem como canal e enlace de comunicação. Este capítulo descreve o modelo de diagnóstico e os principais algoritmos de diagnóstico distribuído em redes de topologia arbitrária.

2.1 Modelo de Diagnóstico

Considere um sistema composto por N nodos conectados por C canais de comunicação. Tanto nodos como canais de comunicação podem estar falhos ou sem-falha. Um algoritmo de diagnóstico permite aos nodos sem-falha do sistema determinar quais nodos estão alcançáveis e quais são inalcançáveis [13]. O modelo de diagnóstico adotado neste trabalho é baseado no modelo PMC [18], cuja principal premissa é que cada nodo é capaz de executar testes em outros nodos e os nodos sem-falha são capazes de determinar corretamente o estado dos nodos testados por eles. Ao testar um nodo sem falha o testador

pode então obter informações de diagnóstico mantidas pelo nodo testado. Desta forma, mensagens de diagnóstico que carregam informações de diagnóstico são propagadas por todo o sistema. Os vértices do chamado *grafo de testes* são os nodo do sistema e uma aresta direcionada representa um teste do nodo testador para o nodo testado.

A figura 2 apresenta um grafo de testes onde as setas indicam uma aresta direcionada do nodo testador para o nodo testado. Assim, a aresta direcionada do nodo 1 para o nodo 2 indica que o nodo 1 testa o nodo 2. Da mesma forma, a aresta direcionada do nodo 2 para o nodo 3 indica que o nodo 2 testa o nodo 3 e assim por diante.

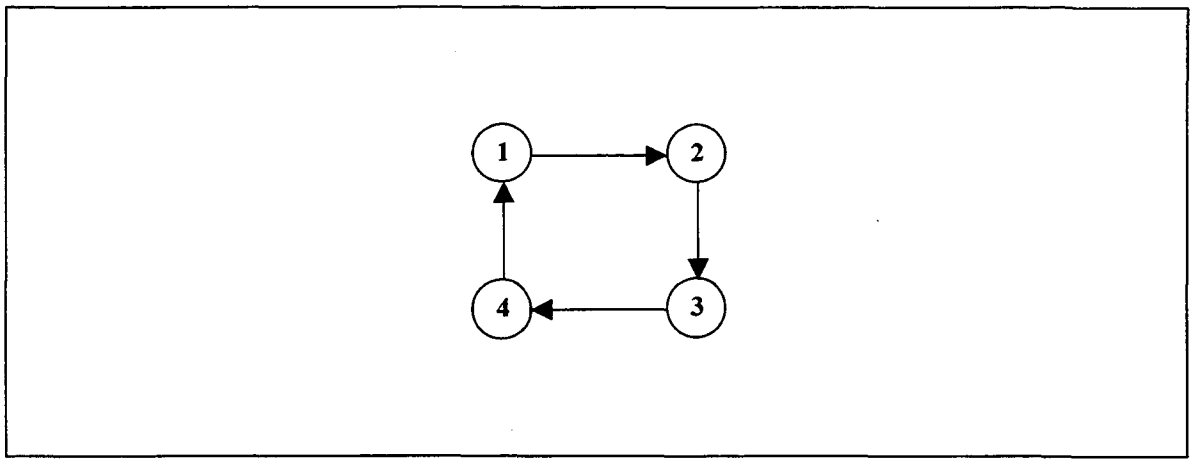


Figura 2: Exemplo de grafo de testes.

Os algoritmos para diagnóstico são executados em *rodadas*. Um nodo executa testes a cada intervalo de testes, que pode ser, por exemplo, 30 segundos. Quando todos os nodos sem-falha terminam de executar os testes previstos para um intervalo, uma rodada de testes se completa. O número de rodadas necessárias para que todos os nodos sem-falha do sistema façam o diagnóstico de um determinado evento é uma medida importante da eficiência de um algoritmo de diagnóstico, e é chamada *latência* do algoritmo.

A seguir é apresentada uma descrição dos principais algoritmos de diagnóstico distribuído de redes de topologia arbitrária.

2.1 O Algoritmo de Bagchi-Hakimi

Bagchi e Hakimi [1] apresentam um algoritmo distribuído para diagnóstico de redes de topologia arbitrária que utiliza o número de mensagens trocadas entre os nodos considerado "ótimo".

Cada nodo pode estar falho ou sem-falha, mas os enlaces de comunicação nunca falham. O algoritmo não é executado on-line; com este algoritmo não é possível implementar um monitoramento dinâmico e contínuo da rede, isto é, durante a sua execução um nodo não pode falhar ou recuperar-se. As falhas são permanentes e ocorrem antes do início da execução do algoritmo.

O algoritmo assume o modelo PMC [18], desta forma um nodo sem-falha sempre pode diagnosticar corretamente qualquer um de seus vizinhos e isto é conseguido através da realização de testes. Baseado nos resultados obtidos nos testes o nodo testador conclui se um dos vizinhos está falho ou sem-falha.

Quando o algoritmo começa, cada nodo sem-falha conhece somente o seu próprio estado e o dos seus vizinhos imediatos. O objetivo do algoritmo é criar uma árvore cujos nodos são todos os nodos sem-falha da rede, ou de cada componente conexo. Desta forma, a partir da raiz da árvore é possível realizar o diagnóstico de todo o sistema: a raiz conhece o estado dos seus filhos, que, por sua vez, conhecem os estados de seus filhos, e assim por diante, até as folhas.

O algoritmo inicia quando nodos espontaneamente despertam e criam suas respectivas árvores, estas árvores vão sendo unidas até que reste apenas uma única árvore. Os estados de todos os nodos são coletados pelo nodo raiz. Em seguida esta informação que foi coletada é enviada, em um único pacote, a todos os nodos da árvore, completando o diagnóstico.

Segundo os próprios autores, o algoritmo é bastante complicado. Se N nodos despertarem e consequentemente N árvores forem criadas, estas árvores passarão pelos processos sucessivos de união, de acordo com regras específicas, de forma que, no final resta apenas uma única árvore na qual ocorre o diagnóstico. Além do fato de não ser executado on-line, outro problema deste algoritmo é em relação ao tamanho das mensagens transferidas entre nodos. Cada nodo ao processar uma mensagem acrescenta o seu identificador, assim tem-se mensagens de tamanho significativo.

2.2 O Algoritmo Adapt

Bianchini, Stahl e Buskens [3, 4] apresentam o algoritmo *Adapt*. O *Adapt* é um algoritmo de diagnóstico distribuído de redes de topologia arbitrária executado on-line, isto é um nodo da rede pode falhar ou recuperar-se durante a execução do algoritmo. O algoritmo utiliza o modelo de falhas PMC [18], onde os nodos são capazes de testar outros nodos, assim nodos sem-falha são capazes de avaliar corretamente o estado dos nodos testados, informando resultados de testes confiáveis.

No algoritmo *Adapt* testes são realizados periodicamente. Uma árvore de testes contendo um caminho direcionado de todos os nodos sem-falha para todos os demais nodos da rede é construída. Uma árvore de diagnóstico distribuído é criada pelo algoritmo após a detecção de um evento para disseminar informações de diagnóstico. A seguir o algoritmo é apresentado em detalhes.

2.2.1 Estrutura de Dados Utilizada pelo Algoritmo

Cada nodo da rede tem um identificador único n_i e um vetor *Syndromes* armazenado localmente, contendo uma lista de nodos testados por n_i, bem como o

resultado dos testes realizados. Um contador local, chamado *Timestamp*, também é armazenado indicando a idade da informação.

Um nodo n_i recebe o vetor *Syndromes* de outros nodos através de mensagens de diagnóstico, utilizadas para disseminar a informação entre nodos. Todas as mensagens contém: o identificador do nodo originador da mensagem, ou nodo raiz, uma cópia do vetor *Syndromes* armazenado naquele nodo e uma lista dos nodos que já processaram a mensagem.

2.2.2 As Fases do Algoritmo *Adapt*

O algoritmo está dividido em três fases: *Search*, *Destroy* e *Inform*. Um nodo realiza testes em outros nodos. Quando um evento de falha é detectado pelo nodo testador, este executa o procedimento *Search*, iniciando a primeira fase do algoritmo. Nesta fase todos os nodos vizinhos são examinados e um teste em um vizinho é *adicionado* localmente se ainda não existir caminho para aquele nodo na árvore atual. O procedimento *Search* é executado em paralelo em vários nodos reduzindo a latência de diagnóstico.

A segunda fase, *Destroy*, iniciada após a fase *Search*, remove testes redundantes possivelmente introduzidos na fase *Search*. O procedimento *Destroy* é executado seqüencialmente em cada nodo, onde um teste em um vizinho é removido se existe outros caminhos para aquele vizinho na árvore atual. Embora todos os nodos sem-falha mantenham o diagnóstico correto, eles podem ter seus respectivos vetores *Syndromes* diferentes, armazenados localmente durante a fase *Destroy*. Na terceira fase do algoritmo, um pacote *Inform* é usado para atualizar o vetor *Syndromes* de todos os nodos da árvore atual. Um único pacote *Inform* contendo o vetor *Syndromes*, que está armazenado localmente, é enviado para todos os nodos sem-falha.

2.2.3 Um Exemplo de Execução do Algoritmo

Um exemplo da execução do algoritmo *Adapt* é ilustrado na figura 3. A rede contendo seis nodos é mostrada na figura 3A e a árvore de testes inicial está representada na figura 3B.

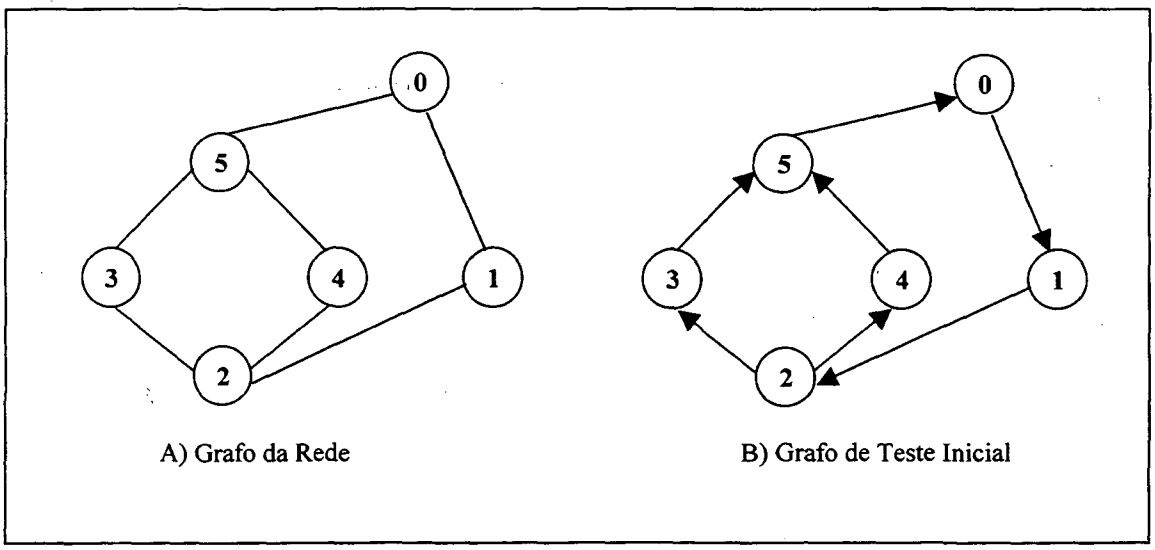
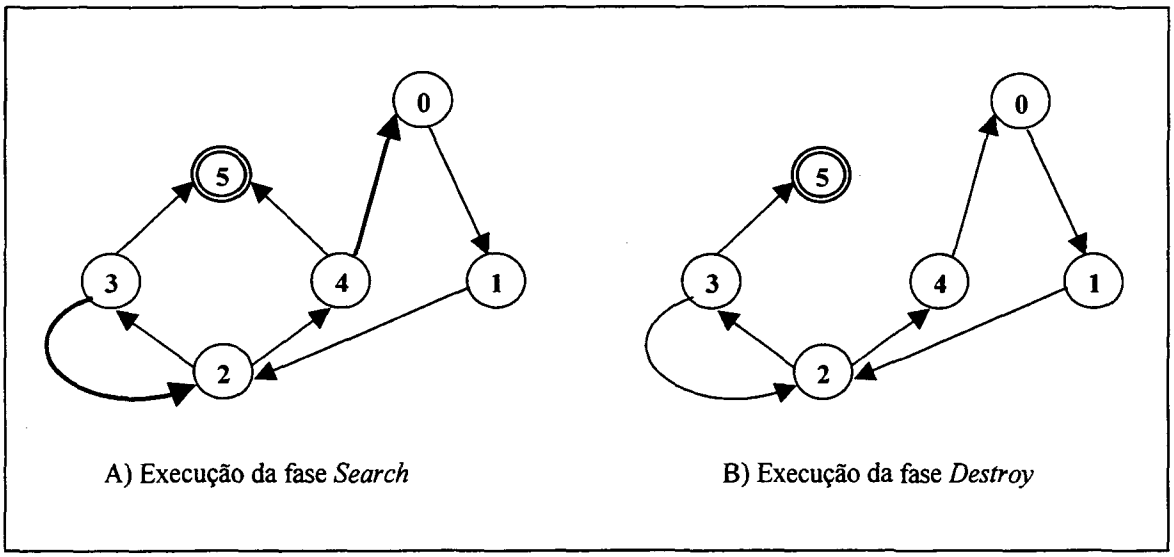


Figura 3: Topologia de rede exemplo e grafo de testes inicial.

Supondo uma falha no nodo 5, o nodo 3 e o nodo 4 detectam a falha e iniciam o procedimento *Search*. O nodo 3 e o nodo 4 adicionam testes pois não existe um caminho de qualquer nodo para qualquer outro nodo na árvore. Mais especificamente o nodo 3 adiciona um teste para o nodo 2 e o nodo 4 adiciona um teste para o nodo 0, conforme mostra a figura 4A. O nodo 3 e o nodo 4 criam um pacote *Search* e o disseminam independentemente para um vizinho sem-falha. Ambos os pacotes são enviados para o nodo 2 e o pacote do nodo 3 chega primeiro. Quando o nodo 2 processa o pacote *Search* vindo do nodo 3 o nodo 2 executa o procedimento *Search* e verifica que ele tem um caminho direto para todos os nodos. O nodo 2 não adiciona testes e dissemina o pacote *Search*.

Na seqüência o pacote *Search* do nodo 4 é recebido pelo nodo 2. O pacote contém o vetor *Syndromes* do nodo 4 atualizado com a falha do nodo 5, enquanto o nodo 2 contém o vetor *Syndromes* atualizado que recebeu do nodo 3. O nodo 2 executa o procedimento *Search,* não adiciona novos testes e inicia um novo pacote *Search,* contendo a informação mais recente de ambos os pacotes. Testes não são adicionados por qualquer um dos nodos uma vez que existe um caminho de todos os nodos para qualquer outros nodos na árvore de testes resultante.. Os dois pacotes circulando colidem em um terceiro nodo, sobrevivendo o pacote com a informação mais recente. Uma vez que novos testes não são adicionados, a fase de *Search* termina quando todos os nodos sem-falha processaram o pacote *Search* armazenando o diagnóstico. A figura 4A ilustra a árvore de testes ao final da fase *Search.*



A) Execução da fase *Search*

B) Execução da fase *Destroy*

Figura 4: Falha do nodo 5.

A fase *Destroy* inicia no nodo onde o pacote *Search* completou o seu percurso, por exemplo o nodo 1. O nodo 1 executa o procedimento *Destroy* e não remove qualquer teste. Ele cria e dissemina um pacote *Destroy* para o nodo 2 que também não remove os testes. O pacote *Destroy* é então enviado para o nodo 3 e o nodo 4, seqüencialmente.

Supondo que o nodo 4 é o primeiro a receber o pacote, então o teste redundante para o nodo 5 é removido. Nenhum outro teste pode ser removido e o pacote visita todos os demais nodos retornando ao nodo que originou o pacote. A figura 4B mostra a árvore de testes resultante. O nodo 1 cria então o pacote *Inform* para distribuir a árvore reconfigurada para todos os nodos sem-falha.

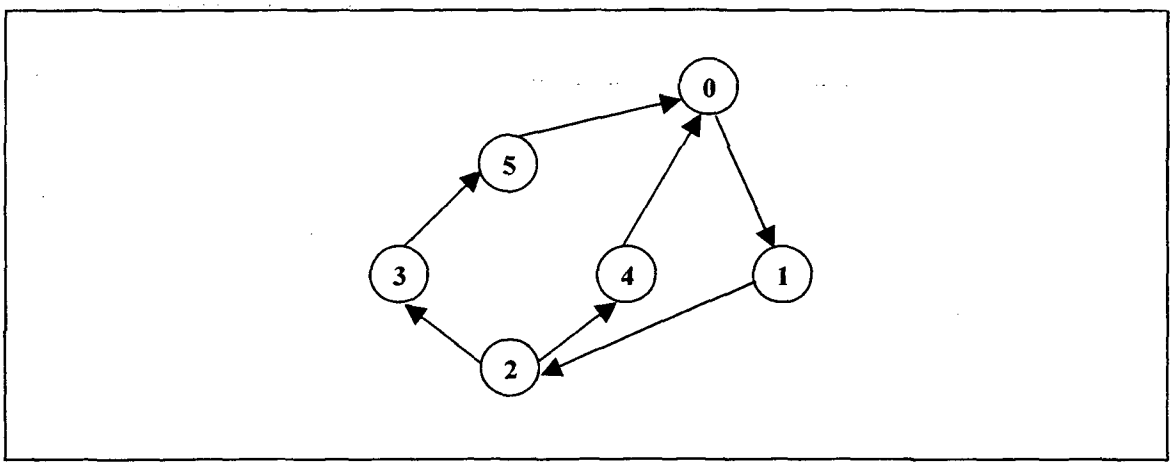


Figura 5: Reparação do nodo 5.

Quando o nodo 5 é reparado, o nodo 3 e o nodo 5 geram pacotes *Search*. O nodo 5 inicialmente testa o nodo 0, determina que ele tem um caminho direto para todos os nodos restantes e não adiciona novos testes. Testes não são adicionados pelos nodos sem-falha restantes uma vez que a árvore de testes contém um caminho direto de um nodo para todos os outros nodos. Quando o procedimento *Search* termina, o procedimento *Destroy* é iniciado por um nodo, por exemplo o nodo 3. O nodo 3 encontra um caminho direto para todos os nodos através do nodo 5, e para de testar o nodo 2. Um pacote *Destroy* é criado e enviado para todos os nodos restantes. A árvore de testes resultante, mostrada na figura 5 difere da topologia de testes inicial, figura 3B. Um pacote *Inform* distribui a árvore de testes reduzida para todos os nodos.

Apesar do algoritmo *Adapt* poder ser executado on-line ele utiliza uma estratégia de disseminação de mensagens seqüencial que foi melhorada nos algoritmos apresentados a seguir.

2.3 O Algoritmo RDZ

Rangarajan, Dahbura e Ziegler [19] apresentam outro algoritmo distribuído para diagnosticar falhas em redes de topologia arbitrária. Este algoritmo é chamado RDZ, das iniciais dos autores.

Neste algoritmo um nodo testa outro nodo periodicamente sendo que cada nodo sem-falha é testado por exatamente um outro nodo. Quando um nodo detecta a falha ou o reparo de um outro nodo que ele estava testando, o nodo testador propaga a nova informação para seus vizinhos sem-falha, os quais propagam a informação para seus vizinhos, e assim por diante. Nodos falhos não são testados, desta forma o *overhead* do sistema monitorado é minimizado durante períodos onde o estado dos nodos da rede não muda. O algoritmo apresenta a menor latência de diagnóstico, sendo esta proporcional ao diâmetro da rede, em função do uso da estratégia de propagação de mensagens em paralelo, novas informações são disseminadas tão rapidamente quanto possível através da rede.

No algoritmo RDZ os nodos detectam falhas em nodos vizinhos e então propagam esta informação para outros nodos na rede. Na detecção, um nodo sem-falha, ao realizar um teste, pode detectar um evento de falha. Quando um evento de falha é detectado por um nodo este executa a disseminação. Na disseminação, o nodo que detectou o evento de falha propaga esta informação para todos os seus vizinhos na rede, os quais propagam a informação para seus vizinhos e assim por diante utilizando a estratégia descrita a seguir.

Uma mensagem de diagnóstico é transmitida de um nodo para seu vizinho por meio de uma *transação de validação*. Quando um nodo *a* propaga a informação para outro nodo *b*, é necessário que o nodo *a* determine se o nodo *b* está falho ou sem-falha. Na transação de validação, se o nodo *a* tem informação a ser enviada para o nodo *b*, ele envia uma mensagem para o nodo *b*, que a processa e envia uma mensagem de confirmação para o nodo *a*. Conforme a mensagem de disseminação vai sendo propagada pela rede, os identificadores do nodos que processam a mensagem vão sendo armazenados na própria mensagem de disseminação.

Um nodo é chamado de *órfão* se nenhum outro nodo está testando-o. Um nodo pode tornar-se um órfão sem-falha, se seu testador falha, ou um órfão falho. Se um nodo sem-falha torna-se falho, seu testador detecta este evento de falha e após disseminar a informação do evento de falha, pára de testar o nodo falho. Assim, todos os nodo falhos, uma vez que eles são detectados por seus testadores, tornam-se órfãos falhos. Quando um nodo falho é reparado, este requisita para um de seus vizinhos que o teste até encontrar um vizinho sem-falha que aceita o seu pedido. O vizinho que reconheceu o pedido de teste inicia a disseminação da informação sobre o evento ocorrido (órfão falho torna-se sem-falha) para os nodos da rede enviando uma transação de validação para todos os seus vizinhos.

Se uma falha ocorrer no testador do nodo sem-falha, então o nodo torna-se um órfão sem-falha porque cada nodo é testado por apenas um outro nodo. Neste caso, o órfão não precisará que um outro nodo o teste imediatamente. Eventualmente um de seus vizinhos sem-falha, se houver um, envia ao órfão uma nova informação sobre o estado dos nodos na rede. Uma vez que isso acontece, o órfão descobre que ele não está sendo testado por ninguém e pede um a um para seus vizinhos que o teste até encontrar um vizinho sem-falha que reconhece o pedido. Um órfão sem falha pode tornar-se falho antes da

informação sobre a falha do seu testador atingi-lo. Se isso acontecer, o evento de falha (órfão sem-falha torna-se um órfão falho) não é detectado imediatamente. Casualmente, o vizinho que enviou a informação sobre o estado da rede para o órfão falho detecta este evento de falha quando ele recebe uma mensagem de confirmação errada do órfão falho ou *time-out* porque ele não recebeu a mensagem de confirmação em tempo, e então inicia a disseminação da informação sobre o evento de falha. Este tipo de configuração é chamada *jellyfish fault configuration* [19], apresentada na figura 6, explicada a seguir.

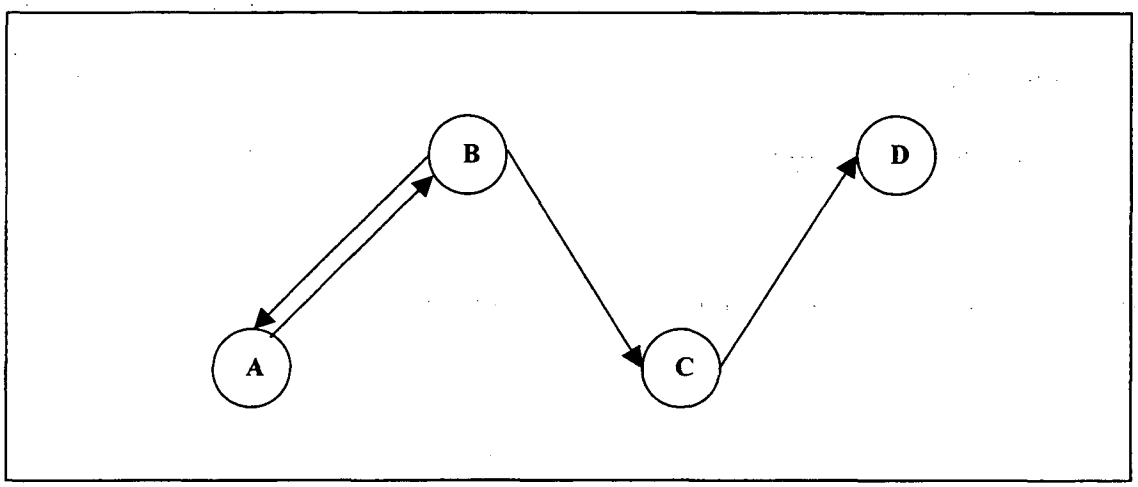


Figura 6: Uma configuração *jellyfish*.

Considere a figura 6, todos os nodos formam uma *jellyfish*, na qual existe um ciclo (nodo A e nodo B) e testes derivam do ciclo, do nodo B para o nodo C para o nodo D. Se ambos os nodos A e B tornarem-se falhos, o nodo C torna-se um órfão sem-falha. Desta forma, os nodos C e D não serão capazes de diagnosticar a falha. O mesmo acontece se os nodos A, B e C tornarem-se falhos, isto é, o nodo D não detecta o evento. Este problema impede o uso do algoritmo RDZ para gerência de falhas de redes.

2.3.1 Um Exemplo de Execução do Algoritmo

Considere a rede da figura 7, de topologia conhecida como $D_{1,2}$, na qual $\forall$ nodo i, $\exists$ arestas $\{(i, i+1), (i, i+2), (i-1,i), (i-2, i)\}$.

Inicialmente a topologia de testes é um anel (o nodo 1 testa o nodo 0, o nodo 2 testa o nodo 1, e assim por diante). Os arcos pontilhados, representados na figura 6, mostram a topologia de testes inicial. A figura 8 apresenta o caminho da mensagem de disseminação, com os tempos nos quais a informação é processada.

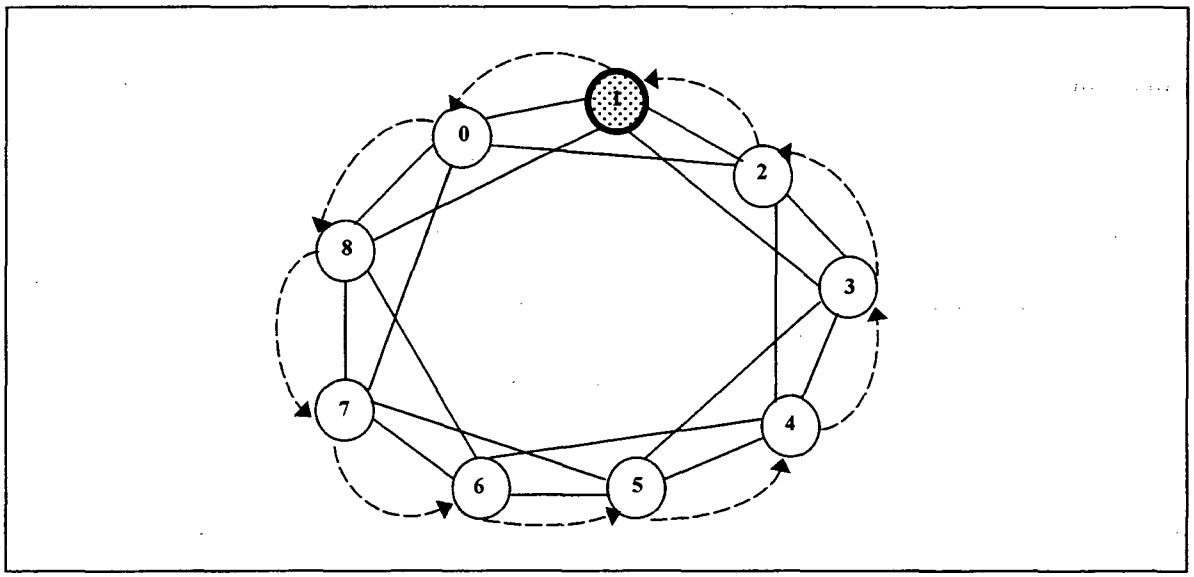


Figura 7: Uma rede de topologia $D_{1,2}$.

Considere que o nodo 1 torna-se falho no tempo 400. O nodo 2 testa o nodo 1 e detecta a falha deste no tempo 1515 (o nodo 2 envia a mensagem de teste no tempo 515 e o período limite de resposta para o teste é configurado para 1000 unidades de tempo). Esta informação é disseminada para todos os seus vizinhos, que por sua vez também disseminam para todos os seus vizinhos e assim sucessivamente, como representado na figura 8. A figura mostra também as mensagens redundantes: o nodo 2 dissemina a mensagem para os seus vizinhos, o nodo 0, o nodo 1, o nodo 3 e o nodo 4; estes também disseminam a mensagem para os seus vizinhos. Por exemplo o nodo 3, dissemina a mensagem para os seus vizinhos, o nodo 1, o nodo 4 e o nodo 5. No entanto a mensagem recebida pelo nodo 4 é redundante, pois ele já conhecia a informação, vinda do nodo 2.

O nodo 7 é o último nodo a aprender sobre a falha do nodo 1 no tempo 1529.5. Isto significa que a latência (o tempo necessário para que todos os nodos sem-falha da rede aprendam sobre um evento de falha uma que vez que ele tenha sido detectado) é 14.5 unidades de tempo para este exemplo.

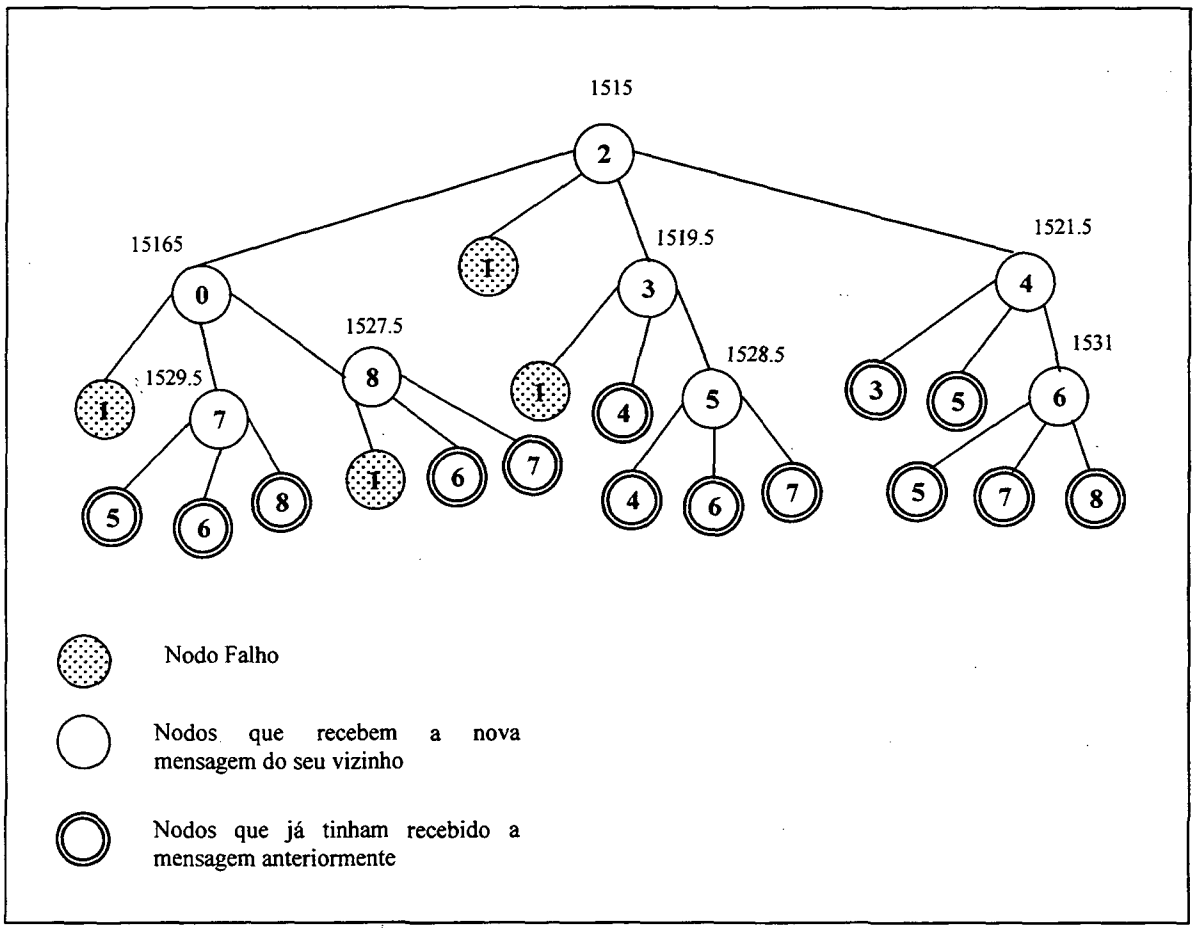


Figura 8: Disseminação das mensagens para o exemplo de execução do algoritmo.

2.4 O Algoritmo NBND

O algoritmo *Non-Broadcast Network Diagnosis* (NBND) [6, 10] foi desenvolvido para ser aplicado em sistemas de gerência de falhas em redes. O algoritmo é totalmente distribuído e tolerante a falhas, pois mesmo que ocorram falhas na rede, os nodos sem falha monitoram a rede continuamente. O algoritmo NBND emprega uma estratégia que permite o diagnóstico de *time-outs* de canais de comunicação e calcula a

conectividade dos nodos da rede usando apenas um teste por canal. A latência do algoritmo é proporcional ao diâmetro da rede.

O algoritmo NBND baseia-se no fato de que é possível determinar se um canal de comunicação está em *time-out* ou se está sem-falha, mas é impossível distinguir se um nodo está falho ou se todos os canais de comunicação que levam ao nodo é que estão falhos. A figura 9 mostra duas configurações ambíguas: 1) a figura 9A ilustra o caso onde todos os canais de comunicação que levam ao nodo estão falhos; 2) a figura 9B ilustra o caso onde um nodo está falho. Assim o algoritmo não calcula quais nodos estão falhos e quais nodos estão sem falha, mas calcula quais nodos estão atingíveis e quais estão inatingíveis. Desta forma, os estados de um canal de comunicação são *sem-falha* e *timed-out* e os estados de um nodo são *sem-falha* ou *inatingível*.

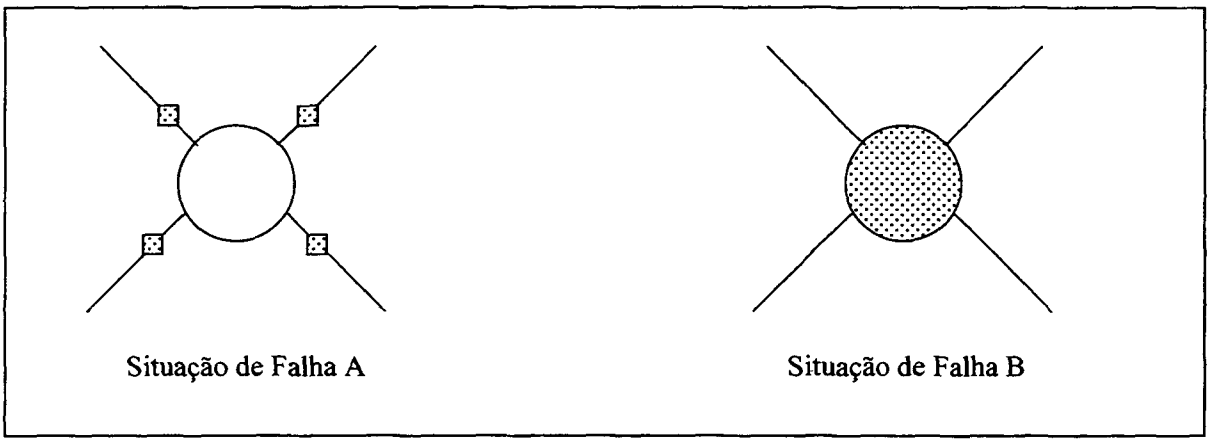


Figura 9: Configurações de falhas ambíguas.

O algoritmo NBND consiste de três etapas: testes dos canais de comunicação, disseminação de informações e diagnóstico da rede.

2.4.1 A Etapa de Testes

Cada canal de comunicação sempre conecta dois nodos, e estes têm identificadores únicos, os nodos com maior identificador testam o canal de comunicação a

cada intervalo de testes. O algoritmo NBND utiliza o menor número de testes, cada canal de comunicação é testado por apenas um dos nodos por ele interligados.

O algoritmo NBND utiliza uma estratégia de testes chamada *two-way testing*. Quando o nodo A testa o canal de comunicação para o nodo B, aquele obtém o tempo local de B e armazena o resultado em B. Assim, não apenas o nodo A sabe sobre o estado de B, mas também o nodo B pode monitorar a atividade do nodo testador A. Quando um tempo limite para o teste do canal de comunicação entre dois nodos é definido e este limite de tempo exceder, então um nodo pode descobrir que o testador está em *time-out*. Quando um nodo detecta o *time-out* de um canal de comunicação ou um nodo testador falha, o nodo inicia ou continua testando o canal de comunicação até terminar o estado de *time-out*. Desta forma quando o canal recupera-se, apenas o nodo de maior identificador testa o canal de comunicação. Esta estratégia de testes garante que a configuração jellyfish, descrita no algoritmo RDZ, sempre seja detectada.

Cada nodo da rede mantém um contador de estados para cada canal de comunicação, o qual é inicializado em zero e incrementado a cada nova informação de evento recebida para aquele canal. Se o canal de comunicação está em *time-out*, isto é, o vizinho não respondeu ao teste, e no intervalo de testes anterior ele havia respondido, então existe um novo evento e este é de falha. Da mesma forma, se o canal de comunicação estava no estado de *time-out* no intervalo de testes anterior, e ele responde ao teste no intervalo de testes atual, então existe um novo evento e este é de reparo. Ao descobrir um evento, de falha ou de reparo, o algoritmo passa para a etapa de disseminação da informação, descrita a seguir.

2.4.2 A Etapa de Disseminação de Informações de Diagnóstico

Quando um evento de falha ou de reparo é descoberto, mensagens de disseminação notificando este evento são propagadas pelos nodos vizinhos. Assim, cada nodo propaga a informação do evento para todos os seus vizinhos sem-falha. Da mesma forma, os vizinhos sem-falha propagam a informação para os seus vizinhos sem-falha e assim por diante. Esta estratégia de disseminação paralela da informação é também usada pelo algoritmo RDZ.

Cada mensagem de diagnóstico carrega informações sobre o identificador do nodo, o contador de estados e qual nodo processou a mensagem, para não precisar reenviá-la. Assim o número de mensagens redundantes é reduzido e as mensagens não ficam circulando pela rede.

Quando chega a mensagem, cada nodo acrescenta seu próprio identificador na lista de nodos que processaram a mensagem. Portanto ele acrescenta o identificador dos vizinhos para os quais a mensagem já foi enviada.

O nodo, após receber a informação sobre o evento de um canal de comunicação, passa para a etapa de diagnóstico para processar a conectividade da rede.

2.4.3 A Etapa de Diagnóstico

Após receber a informação sobre o evento ocorrido no canal de comunicação, o nodo executa um algoritmo, como o algoritmo para a geração da árvore *breadth-first*, para calcular a conectividade da rede, descobrindo quais porções da rede tornaram-se atingíveis ou inatingíveis.

2.4.4 Um Exemplo de Execução do Algoritmo NBND

Um exemplo da execução do algoritmo NBND pode ser acompanhado pela rede apresentada na figura 10. Inicialmente todos os nodos e canais de comunicação estão sem-falha. Cada nodo inicia os testes de acordo com o representado pelas setas, e trocam resultados de teste com os vizinhos. Cada nodo recebe informação sobre todos os canais.

O primeiro evento apresentado acontece quando o canal 3-5 está falho. O evento de *time-out* é detectado pelo nodo 5 e imediatamente disseminado para o nodo 7. O nodo 7 disseminará para o nodo 8 (e este para o nodo 9) e para o nodo 6. O nodo 6 dissemina a informação para o nodo 4 (e este para o nodo 3), nodo 2 e nodo 1. O nodo 2 dissemina a informação para o nodo 3. Se o nodo 3 sair do estado de *timed-out* (canal 3-5) antes da informação chegar do nodo 2, então o nodo 3 também disseminará a informação de *time-out*. Se um nodo, por exemplo o nodo 2, recebe duas mensagens de diagnóstico sobre o mesmo evento ele disseminará somente a primeira delas, a segunda mensagem é reconhecida como informação antiga. Assim, o maior número de mensagens por evento por canal são duas. Após todos os nodos receberem e processarem as mensagens de diagnóstico, estes executam um algoritmo para descobrirem a conectividade da rede e concluírem quais nodos ainda estão conectados.

No segundo evento descrito na figura 10, o nodo 4 torna-se falho. O nodo 6 detecta um *time-out* no canal 6-4 e após terminar o tempo limite de teste, o nodo 2 e o nodo 3 detectam *time-out* nos canais 4-2 e 4-3 respectivamente. Agora, o sistema está dividido em dois componentes conectados formado por:

1) nodo 1, nodo 2 e nodo 3;
2) nodo 5, nodo 6, nodo 7, nodo 8 e nodo 9.

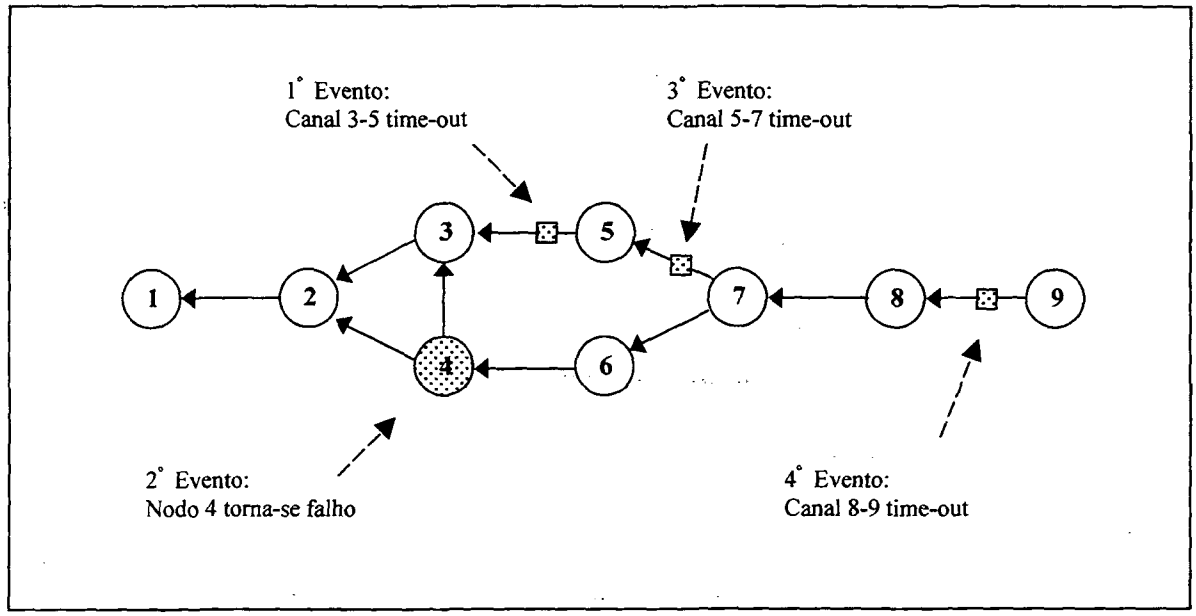


Figura 10: Uma série de eventos ocorre na rede do exemplo.

Como em cada componente um nodo detecta e dissemina o evento, informações de diagnóstico eventualmente atingirão todos os nodos sem-falha na rede.

O terceiro evento ilustrado na figura 10, mostra a falha e *time-out* do canal 7-5. O sistema resultante possui 3 componentes conectados formados por:

1) nodo 1, nodo 2 e nodo 3;

2) nodo5;

3) nodo 6, nodo 7, nodo 8 e nodo 9.

No primeiro componente nenhum nodo detecta o novo evento, porque este já estava desconectado do restante da rede. No segundo componente, o nodo 7 testa o nodo 5 e o enlace está *time-out*; assim o nodo 5 descobre que está desconectado da rede, isto é para todos os outros nodos ele está inatingível. Para o terceiro componente, o nodo 7 detecta que o canal 7-5 está em *time-out* e o evento é disseminado para os outros nodos no componente.

Ainda se o canal 9-8 tornar-se falho e em *time-out*, o nodo 9 detecta o evento e reconhece que está desconectado do sistema. O nodo 8 descobre que o nodo 9 está em *time-out* após terminar o tempo limite de teste, e dissemina a informação do evento para o nodo 7 e para o nodo 6. Os outros nodos na rede já estão em componentes desconectados.

2.4.5 O Impacto do Algoritmo no Desempenho da Rede

Em cada intervalo de teste, cada canal carrega uma mensagem do testador para o vizinho. Além disso, para qualquer novo evento na rede, cada canal carrega normalmente uma e no máximo duas mensagens sobre o evento. A razão é que após atualizar o contador de estados, um nodo não dissemina qualquer outra mensagem que contém informação conhecida. O canal carregará duas mensagens somente se ambos os nodos enviarem informação ao mesmo tempo. Assim o número total de mensagens exigida por evento pelo algoritmo é no máximo 2*C, onde C é o número de canais. Como cada mensagem é pequena, contendo informações sobre um evento e qualquer canal carrega no máximo duas mensagens, o impacto do algoritmo no desempenho da rede é considerado pequeno [10].

2.5 O Algoritmo do Agente Chinês

Duarte e Cestari [7] apresentam um algoritmo de diagnóstico distribuído de redes de topologia arbitrária baseado numa versão distribuída do algoritmo do Carteiro Chinês. O algoritmo do Carteiro Chinês busca a solução de um percurso mínimo em um grafo não euleriano [23], e seu enunciado pode ser assim descrito: dado um grafo qualquer, $G=(V,E)$, achar um caminho que percorra todos os vértices passando por cada aresta ao menos uma vez e realizando o menor percurso possível, sempre retornando ao vértice inicial.

Partindo de um nodo inicial qualquer, um agente móvel, chamado Agente Chinês, realiza seu percurso, determinado pelo algoritmo do Carteiro Chinês, em uma rede de topologia arbitrária, testando os links e disseminando as informações sobre novos eventos. Caso seja detectado um novo evento, a topologia da rede é recalculada e o percurso é reiniciado a partir do nodo que detectou o evento. O percurso é repetido indefinidamente informando todos os nodos sobre o estado de todos os nodos e canais de comunicação. A figura 11 apresenta um exemplo de um percurso pelo agente de diagnóstico.

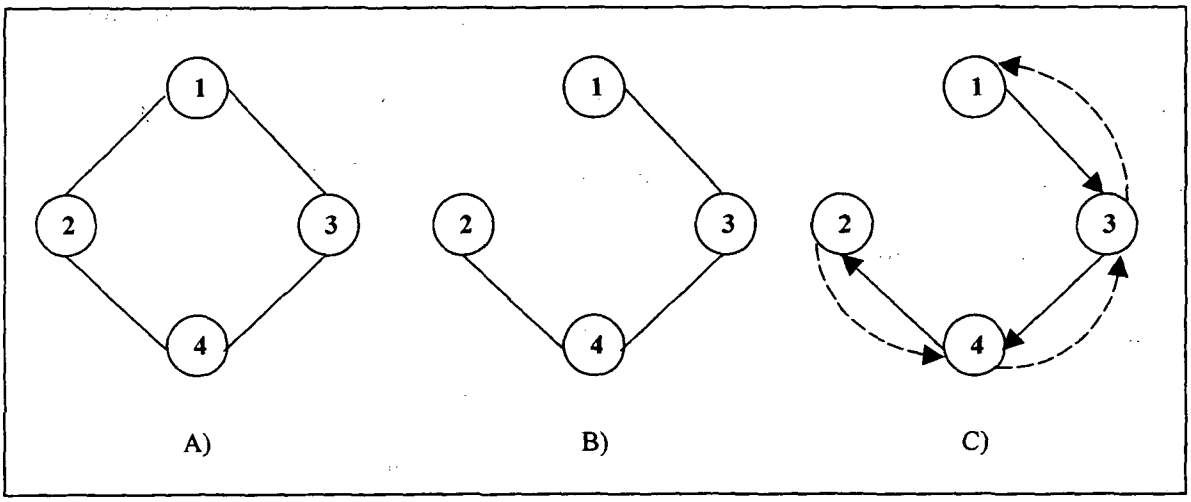


Figura 11: Exemplo de um percurso pelo agente de diagnóstico.

Seguindo o exemplo ilustrado na figura 11A e supondo que ocorra a falha no canal de comunicação 1-2. O percurso da rede começa no nodo 1, sendo o percurso 1, 3, 4, 2 e quando o algoritmo verifica o canal 2-1 descobre que este está falho. Neste momento o Agente Chinês divulga esse novo evento (falha no canal 2-1) a todos os nodos, ou seja, dissemina a informação. No entanto, o grafo resultante da falha é diferente, figura 11B, e um novo percurso é calculado, figura 11C. Após isso, o novo Carteiro Chinês continua seu percurso informando a todos os nodos sobre a falha no canal 2-1, tendo como nodo inicial do percurso o nodo que detectou a falha. O percurso, então é 2, 4, 3, 1, 3, 4, 2, *ou seja*,

todos os nodos da rede receberam a informação que o canal 2-1 está falho. Após isso, o Agente Chinês continua percorrendo o mesmo caminho na rede até que um novo evento seja detectado e o processo é repetido.

A latência do algoritmo é igual ao número de arestas do grafo transformado em euleriano multiplicado pelo intervalo de tempo pré-definido; e o número de mensagens transmitidas na disseminação é igual ao número de arestas do grafo euleriano. A ocorrência de mensagens redundantes não ocasiona um impacto significativo na rede.

O Agente Chinês não permite a ocorrência de eventos dinâmicos, ou seja, a detecção e a recuperação de eventos simultâneos, bem como o particionamento da rede. Entretanto as etapas de teste e disseminação são seqüenciais, desta forma a latência do algoritmo é maior que a dos algoritmos apresentados anteriormente.

O próximo capítulo apresenta uma descrição detalhada do novo algoritmo distribuído de diagnóstico de redes de topologia arbitrária proposto neste trabalho.

Capítulo 3

Um Algoritmo de Diagnóstico de Redes de Topologia Arbitrária Baseado em Inundação de Mensagens

Neste capítulo o novo algoritmo distribuído de diagnóstico de redes de topologia arbitrária [8] é descrito.

Considere um sistema distribuído formado por N unidades, cada uma tendo um identificador único, conectadas por uma rede de topologia arbitrária. A rede, também chamada de sistema, é representada por um grafo não direcionado $G = (V,E)$, sendo V o conjunto de todos os vértices e E o conjunto de todas as arestas do grafo. Cada vértice $u \in V$ equivale a um nodo, e uma aresta $(u, v) \in E$ representa um enlace de comunicação entre os vértices u e v. Um nodo u é vizinho de um outro nodo v se existe uma única aresta (u, v) entre eles em G.

O algoritmo consiste de três etapas: *teste*, *disseminação* e *diagnóstico*. Na etapa de teste os nodos sem-falha testam os enlaces aos quais estão conectados. Casualmente, *eventos* são detectados. Um evento é definido como sendo a mudança de estado de *sem-falha* para o estado de *falha* ou vice-versa tanto de enlaces como de nodos.

Na etapa de disseminação, informações sobre novos eventos detectados na etapa de teste são disseminados paralelamente pela rede, através de um algoritmo baseado em inundação de mensagens. Mensagens redundantes podem ser geradas durante a etapa de disseminação, sendo que um mecanismo para descartar estas mensagens redundantes é utilizado. Na etapa de diagnóstico, a conectividade da rede sob o ponto de vista dos nodos sem-falha é calculada.

A principal contribuição do novo algoritmo em comparação a outros similares já publicados é sua estratégia de disseminação de mensagens. O uso da estratégia de disseminação de mensagens em paralelo reduz o tempo necessário para que todos os nodos fiquem sabendo que ocorreu um evento na rede. Comparando com abordagens baseadas em árvores de disseminação [10, 19], a estratégia baseada em inundação é simples, tolera novos eventos de falha que possam ocorrer e não apresenta impacto significativo no desempenho da rede, pois o tamanho da mensagem de disseminação utilizada é pequeno. O algoritmo não trabalha com eventos dinâmicos, ou seja, não pode ocorrer um novo evento antes do diagnóstico do evento anterior ter sido concluído. Assume-se que um evento de falha não particiona a rede.

A seguir são apresentados o modelo de diagnóstico utilizado, bem como a etapa de teste, a etapa de disseminação e a etapa de diagnóstico.

3.1 Modelo de Diagnóstico

O modelo de diagnóstico usado pelo algoritmo é o modelo PMC [18], proposto por Preparata, Metze e Chien, cuja principal premissa é que nodos sem-falha são capazes de avaliar corretamente a condição (sem-falha ou falho) dos outros nodos que eles testam, enquanto os nodos falhos reportam resultados indefinidos.

Para a implementação dos testes, diversas abordagens já foram consideradas, por exemplo:

- a incorporação de *checkpoints* no projeto de hardware de cada nodo [14]. Assim, um nodo *u* ao testar um outro nodo *v* pergunta pelo valor dos *checkpoints*. O nodo *u* pode diagnosticar o nodo *v* comparando os valores recebidos com seus próprios valores;
- cada nodo possui um conjunto de perguntas em software bem como as respostas às perguntas. Durante o teste, um nodo *u* envia suas perguntas para um outro nodo *v*, o qual devolve as suas respostas. O nodo *u* pode realizar o diagnóstico do nodo *v* comparando as informações recebidas com as suas próprias respostas.

Da mesma forma que o algoritmo NBND, o algoritmo proposto emprega uma estratégia que permite o diagnóstico de *time-outs* dos enlaces. O algoritmo baseia-se no fato de que é possível determinar se um enlace está em *time-out* ou se está sem-falha, mas é impossível distinguir se um nodo está falho ou se todos os enlaces que levam ao nodo é que estão falhos. Para resolver esta ambigüidade, o algoritmo considera que os *estados* de um enlace são *sem-falha* ou *timed-out* e os *estados* de um nodo são *sem-falha* ou *inatingível*.

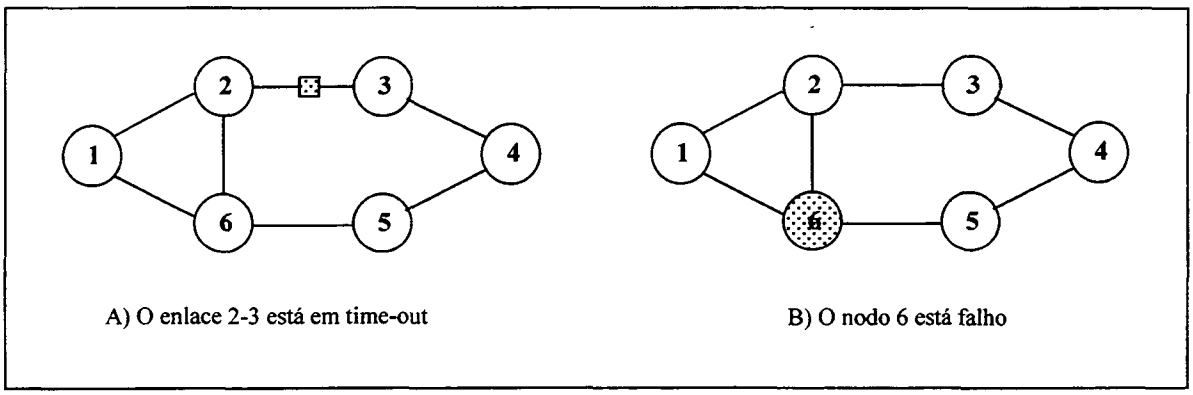


Figura 12: Um exemplo de falhas ambíguas: A) o enlace que conecta o nodo 2 ao nodo 3 está em time-out e B) o nodo 6 está falho.

A figura 12 mostra duas situações de falhas ambíguas: a) o enlace que conecta o nodo 2 ao nodo 3 está falho, assim quando o nodo 3 testa o nodo 2 esta situação é indistinguível para o testador pois este não consegue determinar se o nodo 2 está falho ou se o enlace para o nodo 2 é que está falho; b) o nodo 6 está falho, assim o nodo 1 e o nodo 5 ao testarem o nodo 6 não conseguem distinguir se o nodo 6 está falho ou se o enlace é que está falho.

3.2 Etapa de Testes

A primeira etapa do algoritmo é a de *testes*. Nesta etapa é necessário que cada nodo teste todos os enlaces aos quais está conectado. Os testes são realizados em intervalos de tempo definidos de acordo com o sistema. Em geral o procedimento que implementa o teste depende da tecnologia do sistema. Para otimizar o número de testes executados o algoritmo pode empregar as estratégias propostas para o algoritmo NBND: *two-way-testing* e *token-based*, descritas a seguir.

Na estratégia de testes *two-way-testing* [10] apenas um dos nodos que se conecta ao enlace executa o teste, por exemplo aquele de maior identificador. O nodo u ao testar o enlace para o nodo v obtém o tempo local de v armazenando o resultado em v. Assim, o nodo v também monitora a atividade do nodo testador u. Quando o tempo limite para o teste do enlace é excedido, o nodo testado pode descobrir que o enlace que o conecta ao nodo testador está em *time-out*. Um nodo ao detectar o *time-out* de um enlace ou a falha de um nodo testador, inicia o teste ou continua testando o enlace até terminar o estado de *timed-out*. Desta forma quando o enlace recupera-se, apenas o nodo de maior identificador continua testando o enlace.

Outra estratégia de testes que pode ser utilizada pelo novo algoritmo é a estratégia baseada em passagem de bastão [22]. Na estratégia de testes baseada em

passagem de bastão, cada par de vizinhos carrega um bastão. Em um intervalo de testes, somente um nodo de cada par tem o bastão. Este nodo é chamado de testador e executará um teste no outro nodo, chamado de nodo testado, no próximo intervalo de testes. Após o teste, o nodo testado obtém o bastão e os nodos invertem os papéis. Esta estratégia distribui igualmente a execução do teste entre todos os nodos. Além disso, a estratégia reduz o tempo médio na detecção de um novo evento, apresentando menor impacto na fase de disseminação, exigindo menos mensagens quando comparada com a estratégia *two-way-testing*.

3.3 Disseminação de Informações Sobre Eventos: Um Algoritmo Baseado em Inundação de Mensagens

A segunda etapa do algoritmo é a *disseminação*. Nesta etapa, quando um evento é detectado, mensagens de disseminação notificando este evento são propagadas paralelamente através dos nodos vizinhos, de tal forma que, a partir de sua origem (o nodo que detectou o evento), todos os nodos sem falha sejam informados do evento ocorrido.

No algoritmo proposto, o nodo que descobriu o evento cria uma mensagem de disseminação e a envia para todos os seus vizinhos sem-falha, os quais, por sua vez, também a enviam para todos os seus vizinhos, exceto para o(s) nodo(s) de onde veio a mensagem, e assim por diante. Este processo de disseminação de informações é chamado inundação ou *flooding* [25].

A mensagem de disseminação é composta por três campos: 1) o identificador do nodo testador, 2) o identificador do nodo testado e 3) um contador de eventos, como apresenta a figura 13A. O primeiro campo da mensagem, é o identificador do nodo testador. Este campo armazena o identificador do nodo que está testando o enlace. No

segundo campo da mensagem, o identificador do nodo testado, ou seja o identificador do segundo nodo que é conectado ao enlace sendo testado, é armazenado.

No terceiro campo da mensagem é armazenado um contador de eventos, inicializado em zero, indicando o estado inicial sem-falha. A cada novo evento ocorrido no enlace, o contador é incrementado de uma unidade, sendo possível determinar para cada enlace o número de eventos ocorridos. Um valor par do contador indica que o enlace está sem-falha; um valor ímpar indica que o enlace está falho.

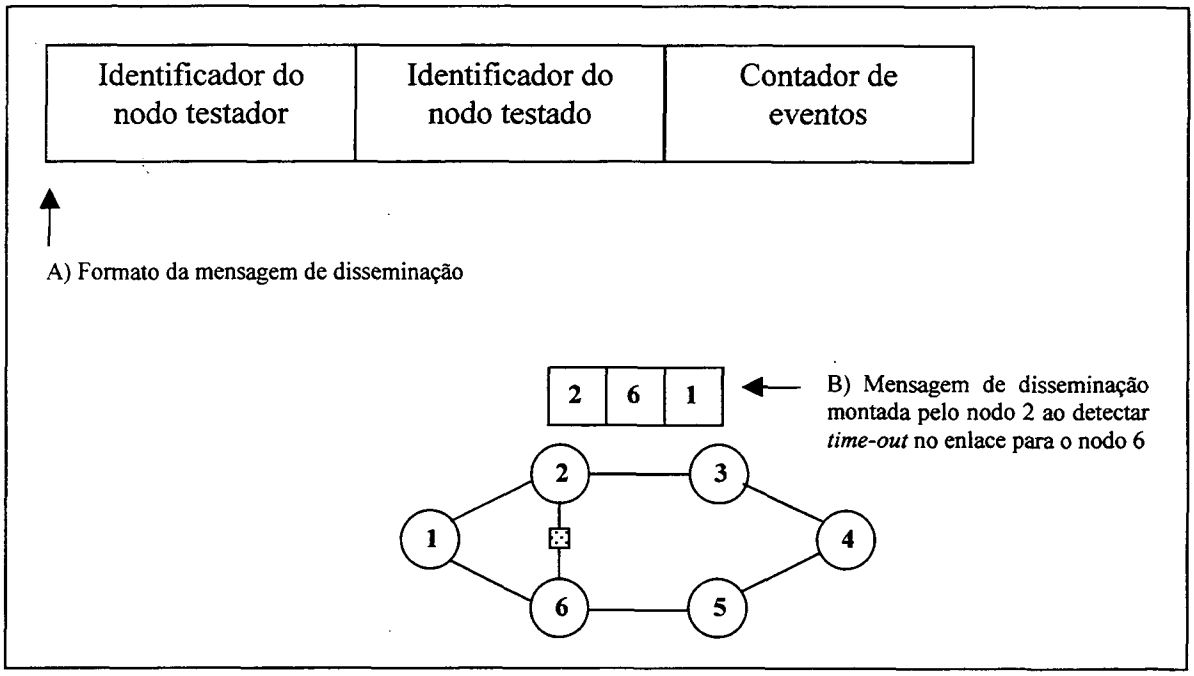


Figura 13: Formato da mensagem de disseminação e exemplo de mensagem de disseminação enviada aos nodos sem-falha da rede.

Considere o exemplo apresentado na figura 13B. Em um intervalo de testes o nodo 2 testa o enlace para o nodo 6 e detecta o evento de falha. Na seqüência, o nodo 2 monta a mensagem de disseminação que será enviada aos nodos sem-falha do sistema. A mensagem é composta pelos seguintes campos: 1) identificador do nodo testador, que neste caso é o nodo 2; o identificador do nodo testado, que neste exemplo é o nodo 6 e 3) o

contador de eventos no enlace entre o nodo testador e o nodo testado, neste caso 1 pois trata-se do primeiro evento.

Devido ao fato do algoritmo utilizar uma estratégia de disseminação em paralelo, mensagens redundantes são geradas. Um mecanismo para descartar as mensagens redundantes é utilizado. Um nodo, ao receber uma mensagem de disseminação, utiliza o contador de eventos para comparar as informações contidas na mensagem com as suas próprias informações. Se as informações recebidas já forem conhecidas pelo nodo que as recebe, o nodo ignora a mensagem, caso contrário ele atualiza as suas informações locais de diagnóstico, e dissemina a mensagem para seus vizinhos, exceto para aquele(s) de onde a mensagem foi recebida.

A figura 14 apresenta um exemplo do mecanismo de descarte de mensagens redundantes implementada pelo algoritmo que está sendo apresentado. O nodo 2 ao testar o nodo 6 detecta a falha no enlace que os conecta e monta a mensagem de disseminação enviando-a aos seus vizinhos, o nodo 1 e o nodo 3. O nodo 1 e o nodo 3 ao receberem a mensagem comparam-na com as suas informações verificando que ainda não conhecem as informações contidas na mensagem, atualizam as suas informações e também disseminam para os seus vizinhos. O nodo 1 dissemina para o nodo 6 e o nodo 3 dissemina para o nodo 4. O nodo 6 o nodo 4 realizam o mesmo procedimento, disseminando a mensagem para os seus vizinhos. O nodo 6 dissemina a mensagem para o nodo 5 e o nodo 4 também dissemina para o nodo 5. Supondo que a mensagem tenha chegado no nodo 5 vinda primeiramente do nodo 6, o nodo 5 processa a mensagem e dissemina para os seus vizinhos, neste exemplo o nodo 4. O nodo 4 processa a mensagem e verifica que a informação contida na mensagem já é conhecida por ele e descarta a mensagem redundante. Quando a segunda mensagem chega no nodo 5, vinda do nodo 4, ele verifica

que a mensagem é redundante descartando-a. Neste exemplo duas mensagens redundantes foram geradas pelo algoritmo.

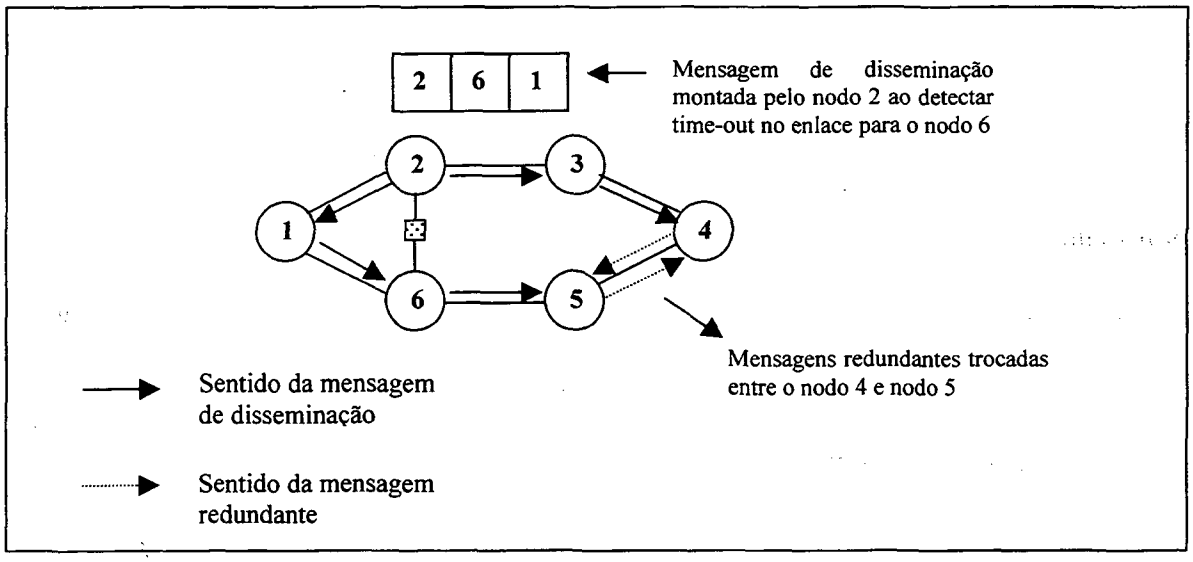


Figura 14: Exemplo de mensagens redundantes.

3.4 Etapa de Diagnóstico

Usando as informações armazenadas sobre o estado dos enlaces, um nodo sem-falha executa um algoritmo para determinar a conectividade do componente ao qual pertence. Esta etapa é executada de forma idêntica mesmo na presença de particionamento da rede (não previsto nas etapas anteriores).

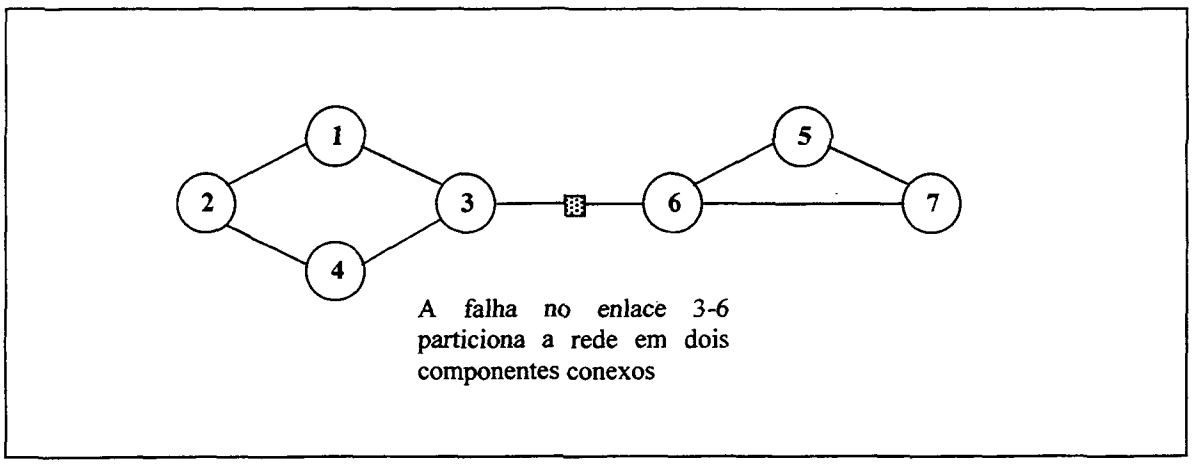


Figura 15: Um exemplo de rede particionada.

Um exemplo pode ser observado na figura 15 onde uma falha no enlace 3-6 particiona a rede. Se o nodo 3, por exemplo, calcula a conectividade do grafo usando um algoritmo como o da árvore *breadth-first* [24], ele determina que o componente conexo ao qual pertence contém o nodo 1, nodo 2, nodo 3 e nodo 4. Outro componente conexo é formado pelo nodo 5, nodo 6 e nodo 7. O algoritmo é executado normalmente em ambos os componentes conexos.

3.5 O Algoritmo

O algoritmo é apresentado na figura 16 utilizando a notação CSP *(Communicating Sequential Processes)* [13].

```
Início /* Algoritmo executado no nodo u */
      Faça para sempre
            Teste: Para cada link u-v que conecta o nodo u ao nodo v
                        Testa link u-v
                        Usando o resultado do teste efetuado em v
                        Se o estado do link mudou
                              Atualizar informações locais
                              Montar a mensagem de disseminação
                              Iniciar o evento disseminação
                   Fim-para
                   Dormir até o início do próximo intervalo de testes

            Disseminação: /* Chega mensagem de vizinho ou evento é
                             detectado */
                  Para cada vizinho v de u
                        Se a informação recebida não é conhecida por v
                              Enviar a mensagem de disseminação para v

            Diagnóstico:
                  Para cada nodo v
                        Calcular a conectividade do grafo considerando
                        as informações correntes de diagnóstico
Fim.
```

Figura 16: O algoritmo em pseudo-código.

No próximo capítulo são apresentados resultados experimentais da execução do algoritmo em redes de diversas topologias, bem como a sua comparação com outros algoritmos funcionalmente equivalentes.

Capítulo 4

Resultados de Simulação

Neste capítulo são apresentados resultados experimentais obtidos através da simulação do algoritmo de diagnóstico de redes de topologia arbitrária descrito no capítulo 3. O algoritmo foi executado em diversos tipos de topologias, dentre elas uma topologia exemplo com 7 vértices cujos graus variam de 2 a 4, a topologia $D_{1,2}$ com 9 vértices de grau 4, topologias hipercubos de 16, 64 e 128 vértices, além de topologias randômicas e a topologia da Rede Nacional de Pesquisa (RNP) [20]. Tais topologias foram selecionadas devido ao fato de já terem sido simuladas em outros algoritmos, como o NBND e o algoritmo do Agente Chinês, desta forma os resultados puderam ser comparados.

Foram simuladas falhas de um enlace em cada grafo, sendo que os dois nodos nos quais o enlace incide detectam o evento em instantes de tempo distintos e disseminam mensagens distintas. No final do capítulo uma seção é dedicada à comparação de resultados obtidos através de simulação de outros algoritmos.

A biblioteca de simulação de eventos discretos *SiMulation Programming Language* (SMPL) [16] foi utilizada. Os nodos foram modelados como *facilities* da SMPL. Nestas simulações o intervalo de testes convencionado foi de 30 unidades de tempo e o intervalo de disseminação de mensagens foi de 1 unidade de tempo. Tais valores foram

escolhidos por terem sido amplamente utilizados em simulações de outros algoritmos como o RDZ, NBND e o Agente Chinês. A seguir são apresentada as descrições das simulações e os resultados obtidos.

4.1 Uma Topologia Exemplo

Considere o grafo da topologia exemplo apresentado na figura 17. O algoritmo inicia a sua execução em t=0,0, o enlace entre o nodo 1 e o nodo 3 torna-se falho no tempo t=20,0. Na primeira rodada de testes todos os nodos testam o enlace para seus vizinhos, considerando dois nodos conectados por um enlace. Assim, o nodo 1 testa o enlace para o nodo 2, nodo 5, nodo 4 e nodo 3; o nodo 2 testa o enlace para o nodo 1 e nodo 5; o nodo 5 testa o enlace para o nodo 1, nodo 2, nodo 4 e nodo 7; o nodo 4 testa o enlace para o nodo 1, nodo 3, nodo 5 e nodo 6; o nodo 3 testa o enlace para o nodo 1, nodo 4 e nodo 6; o nodo 6 testa o enlace para o nodo 3, nodo 4 e nodo 7; o nodo 7 testa o enlace para o nodo 5 e nodo 6. Como neste intervalo de testes os nodos testadores não detectam um novo evento, os testes são reiniciados em t=30,0. No instante t=20,0 o enlace entre o nodo 1 e o nodo 3 torna-se falho. Em t=30,0 o nodo 1, ao testar o enlace para o nodo 2, nodo 5, nodo 4, e nodo 3 detecta a falha no enlace que o conecta ao nodo 3. Neste momento, o nodo 1 prepara-se para iniciar o processo de disseminação de mensagens, de acordo com o novo evento detectado, formando então a mensagem de disseminação. Esta contém o identificador do nodo testador igual a 1, o identificador do nodo testado igual a 3 e o contador igual a 1, conforme ilustrado na figura 17A.

O nodo 1 inicia o evento de disseminação da mensagem para o nodo 2, nodo 5 e nodo 4 em t=31,0. O nodo 2, nodo 5 e nodo 4 ao recebê-la, comparam as suas informações locais de diagnóstico com as informações contidas na mensagem. Verificam que as informações ainda não são conhecidas por eles, atualizam as suas informações

locais de diagnóstico e disseminam a mensagem para os seus vizinhos. Em t=32,0 o nodo 2 dissemina a mensagem para o nodo 5; o nodo 5 dissemina a mensagem para o nodo 2, nodo 4 e nodo 7; o nodo 4 dissemina a mensagem para o nodo 3, nodo 5 e nodo 6. O nodo 2 e o nodo 4 ao receberem a mensagem vinda do nodo 5 comparam as informações contidas na mensagem com as suas informações locais verificando que as informações já são conhecidas, estas duas mensagens redundantes são descartadas. O mesmo acontece com o nodo 5 ao receber as mensagens vindas do nodo 2 e nodo 4, este já tem conhecimento sobre as informações contidas nas mensagens e descarta estas duas mensagens redundantes.

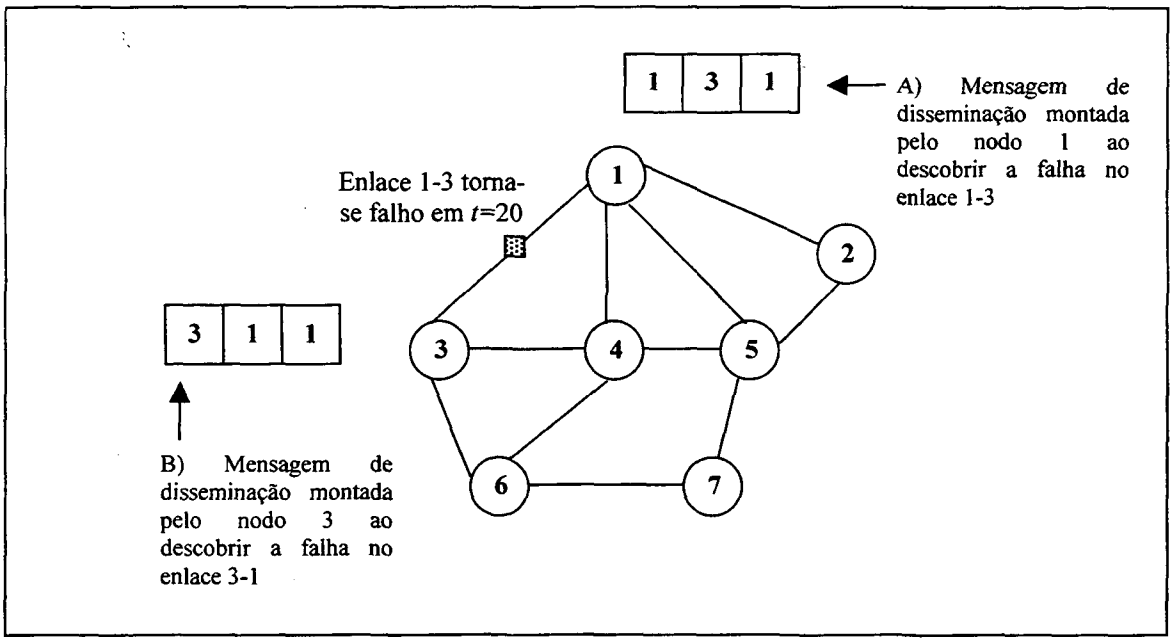


Figura 17: Grafo da topologia exemplo.

Em t=33,0 o nodo 6, nodo 7 e nodo 3 disseminam a mensagem. O nodo 6 dissemina a mensagem para o nodo 7 e nodo 3; o nodo 7 dissemina a mensagem para o nodo 6; o nodo 3 dissemina a mensagem para o nodo 6. O nodo 6 ao receber as mensagens, vindas do nodo 7 e nodo 3 descarta-as pois são redundantes. O nodo 7 e o nodo 3 também descartam a mensagem redundante vinda do nodo 6. O nodo 3 ao tentar disseminar a

mensagem para o nodo 1 detecta a falha no enlace que o conecta ao nodo 1. A mensagem de disseminação montada pelo nodo 3 é apresentada na figura 17B.

Em t=34,0 o nodo 3 dissemina a nova mensagem para o nodo 4 e nodo 6. O nodo 4 e o nodo 6 ao receberem a mensagem comparam as suas informações locais de diagnóstico com as informações contidas na mensagem. Detectam que a informação não é conhecida por eles, atualizando então as suas informações locais, e disseminam a mensagem para os seus vizinhos. Em t=35,0 o nodo 6 dissemina a mensagem para o nodo 4 e nodo 7; o nodo 4 dissemina a mensagem para o nodo 6, nodo 5 e nodo 1. O nodo 4 e o nodo 6 descartam as mensagens redundantes trocadas entre eles. Em t=36,0 o nodo 7 dissemina a mensagem para o nodo 5; o nodo 5 dissemina a mensagem para o nodo 7, nodo 2 e nodo 1; o nodo 1 dissemina a mensagem para o nodo 5 e nodo 2. As mensagens trocadas entre o nodo 5 e o nodo 7 são redundantes e descartadas. O mesmo acontece com o nodo 1 e o nodo 5. Em t=37,0 o nodo 2 dissemina a mensagem para o nodo 1, sendo descartada por ser redundante. O algoritmo termina sua execução quando todos os nodos processaram e disseminaram a mensagem.

Em resumo os resultados são apresentados na tabela abaixo:

Total de mensagens de diagnóstico	**Mensagens redundantes**	**Tempo total para completar o diagnóstico (em unidades de tempo)**
28	16	7

Neste exemplo o número total de mensagens de diagnóstico na rede foi 28, sendo 16 mensagens redundantes. O tempo total para completar o diagnóstico foi de 7 unidades de tempo.

4.2 Topologia $D_{1,2}$

A figura 18 apresenta uma rede de topologia $D_{1,2}$ com nove vértices de grau 4, numerados seqüencialmente, possuindo arestas para os dois vértices posteriores e os dois vértices anteriores na seqüência. A execução do algoritmo inicia em t=0,0, o enlace entre o nodo 6 e o nodo 8 torna-se falho no tempo t=20,0. Na primeira rodada de testes todos os nodos testam o enlace para seus vizinhos. Desta forma o nodo 1 testa o enlace para o nodo 2, nodo 3, nodo 8 e nodo 9; o nodo 2 testa o enlace para o nodo 1, nodo 3, nodo 4 e nodo 9; o nodo 3 testa o enlace para o nodo 1, nodo 2, nodo 4 e nodo 5, e assim por diante. Como neste intervalo de testes os nodos testadores não detectam um novo evento, os testes são reiniciados em t=30,0, sendo que no instante t=20,0 o enlace entre o nodo 6 e o nodo 8 torna-se falho. Em t=30,0, o nodo 6 ao testar o enlace para o nodo 4, nodo 5, nodo 7 e nodo 8 detecta a falha no enlace que o conecta ao nodo 8. Neste momento o nodo 6 monta a mensagem de disseminação que será enviada aos nodos da rede. A mensagem é apresentada na figura 18A.

Em t=31,0 o nodo 6 inicia o processo de disseminação da mensagem enviando-a aos seus vizinhos: o nodo 4, nodo 5 e nodo 7. O nodo 4, nodo 5 e nodo 7 comparam as informações contidas na mensagem com as suas próprias informações de diagnóstico, verificam que ainda não conhecem a informação, processam-na e disseminam para os seus vizinhos. Em t=32,0 o nodo 4, nodo 5 e nodo 7 disseminam a mensagem para os seus vizinhos. O nodo 4 dissemina a mensagem para o nodo 3, nodo 5 (redundante) e nodo 2. O nodo 5 dissemina a mensagem para o nodo 4 (redundante), nodo 3 (redundante) e nodo 7 (redundante). O nodo 7 dissemina a mensagem para o nodo 5 (redundante), nodo 8 e nodo 9. Estes nodos processam a mensagem e disseminam para os seus vizinhos em t=33,0.

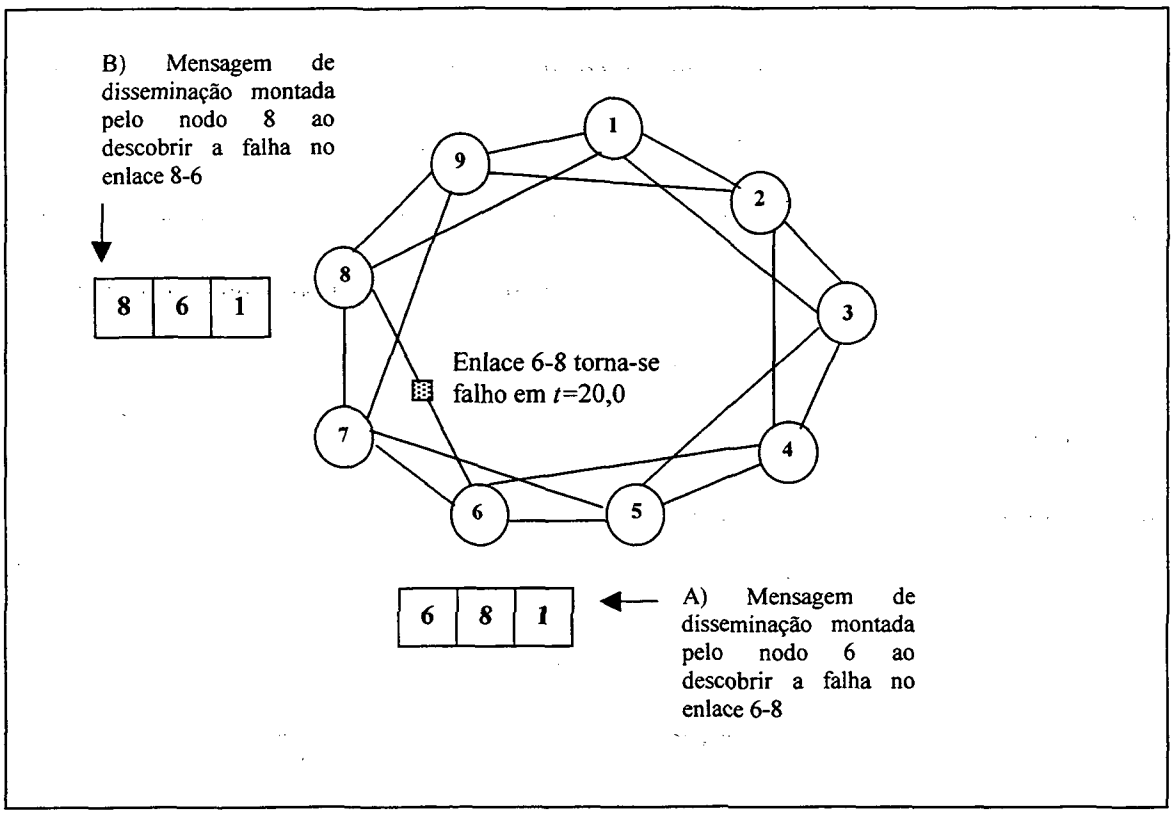


Figura 18: Uma rede de topologia $D_{1,2}$ com 9 nodos.

Em t=33,0 o nodo 3 dissemina a mensagem para o nodo 2 (redundante), nodo 1 e nodo 5 (redundante). O nodo 2 dissemina a mensagem para o nodo 1 (redundante), nodo 3 (redundante) e nodo 9 (redundante). O nodo 9 dissemina a mensagem para o nodo 8 (redundante), nodo 1 (redundante) e nodo 2 (redundante). O nodo 8 dissemina a mensagem para o nodo 9 (redundante) e nodo 1 (redundante). Ao tentar disseminar a mensagem para o nodo 6, o nodo 8 detecta a falha no enlace que o conecta ao nodo 6, montando a mensagem de disseminação que será disseminada aos nodos da rede como mostra a figura 18B.

Em t=34,0 o nodo 1 dissemina a mensagem para o nodo 2 (redundante), nodo 8 (redundante) e nodo 9 (redundante). O nodo 8 inicia o processo de disseminação da nova mensagem para os seus vizinhos: o nodo 9, nodo 7 e nodo 1. Estes comparam as informações contidas na mensagem com as suas informações de diagnóstico verificando

que não conhecem a informação. Os nodos processam a mensagem e disseminam para os seus vizinhos. Em t=35,0 o nodo 9 dissemina a mensagem para o nodo 1 (redundante), nodo 2 e nodo 7 (redundante). O nodo 7 dissemina a mensagem para o nodo 9 (redundante), nodo 6 e nodo 5. O nodo 1 dissemina a mensagem para o nodo 9 (redundante), nodo 2 (redundante) e nodo 3. Todos os nodos processam a mensagem e disseminam para os seus vizinhos. Em t=36,0 o nodo 2 dissemina a mensagem para o nodo 1 (redundante), nodo 3 (redundante) e nodo 4. O nodo 6 dissemina a mensagem para o nodo 5 (redundante) e nodo 4 (redundante). O nodo 5 dissemina a mensagem para o nodo 6 (redundante), nodo 3 (redundante) e nodo 4 (redundante). O nodo 3 dissemina a mensagem para o nodo 2 (redundante), nodo 5 (redundante) e nodo 4 (redundante). Todos os nodos processam a mensagem e disseminam para os seus vizinhos. Em t=37,0 o nodo 4 dissemina a mensagem para o nodo 3 (redundante), nodo 6 (redundante) e nodo 5 (redundante). O algoritmo termina quando todos os nodos processaram e disseminaram a mensagem.

Em resumo os resultados são apresentados na tabela abaixo:

Total de mensagens de diagnóstico	**Mensagens redundantes**	**Tempo total para o diagnóstico (em unidades de tempo)**
52	36	7

Neste exemplo o número total de mensagens de diagnóstico na rede foi 52, sendo 36 mensagens redundantes. O tempo total para completar o diagnóstico foi de 7 unidades de tempo. Neste caso o número de mensagens redundantes foi aproximadamente 70% do total de mensagens empregadas pelo algoritmo.

4.3 Topologia Hipercubo

A topologia hipercubo com 16 nodos é apresentada na figura 19. O algoritmo inicia sua execução em t=0,0, o enlace entre o nodo 5 e o nodo 7 torna-se falho no tempo t=20,0. Na primeira rodada de testes todos os nodos testam o enlace para seus vizinhos. Assim o nodo 1 testa o enlace para o nodo 2, nodo 3, nodo 5 e nodo 9; o nodo 2 testa o enlace para o nodo 1, nodo 4, nodo 6 e nodo 10; o nodo 3 testa o enlace para o nodo 1, nodo 4, nodo 7 e nodo 11, a assim sucessivamente. Neste intervalo de testes os nodos testadores não detectam um novo evento e os testes são reiniciados em t=30,0. No instante t=20,0 o enlace entre o nodo 5 e o nodo 7 torna-se falho.

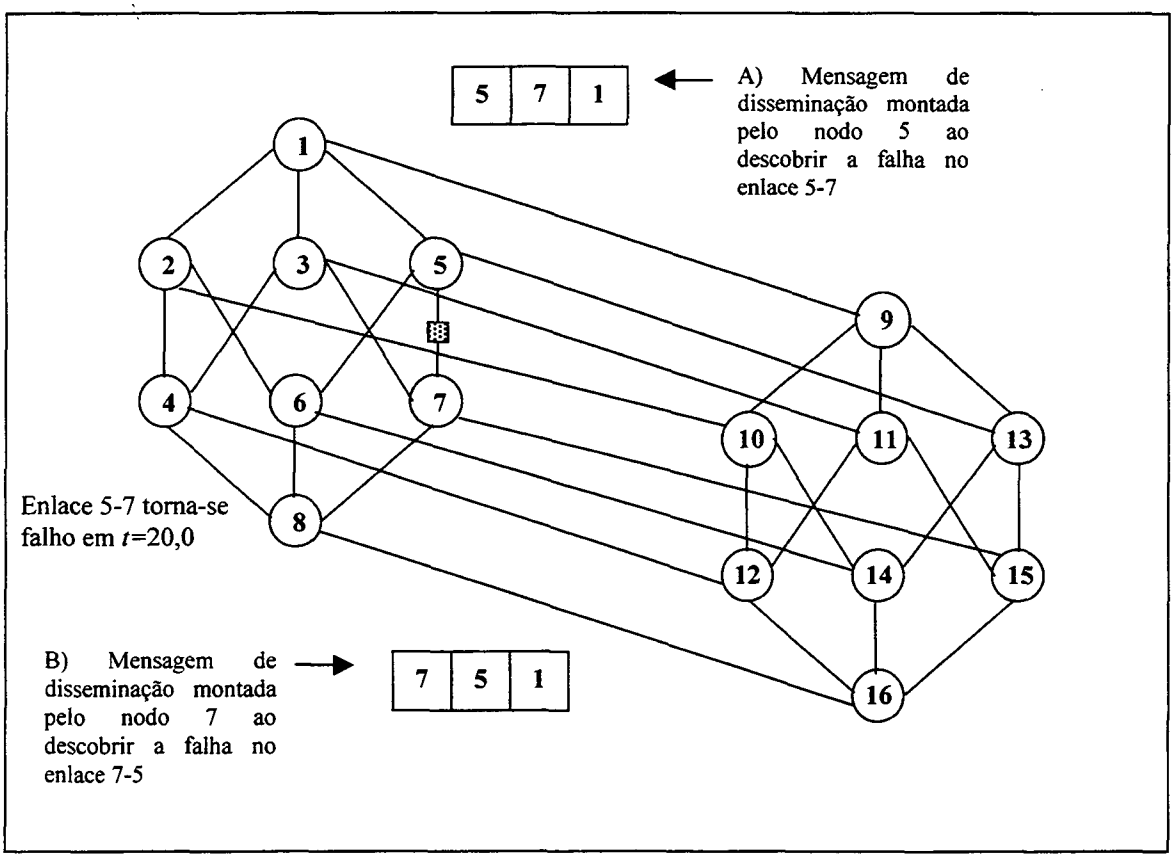


Figura 19: Hipercubo com 16 nodos.

Em t=30,0, o nodo 5 ao testar o enlace para o nodo 1, nodo 6, nodo 7 e nodo 13 detecta a falha no enlace que o conecta ao nodo 7. Neste momento o nodo 5 prepara-se

para iniciar a disseminação da mensagem que será enviada aos nodos da rede. A mensagem de disseminação montada pelo nodo 5 é apresentada na figura 19A.

Em t=31,0 o nodo 5 dissemina a mensagem para os seus vizinhos: o nodo 6, nodo 1 e nodo 13. O nodo 6, nodo 1 e nodo 13 ao receber a mensagem comparam as suas informações com as informações contidas na mensagem, como a informação não é conhecida eles atualizam as suas informações locais e disseminam a mensagem para os seus vizinhos. Em t=32,0 o nodo 6 dissemina a mensagem para o nodo 2, nodo 8 e nodo 14. O nodo 1 dissemina a mensagem para a o nodo 2 (redundante), nodo 3 e nodo 9. O nodo 13 dissemina a mensagem para o nodo 9 (redundante), nodo 14 (redundante) e nodo 15. Todos os nodos processam a mensagem e disseminam para os seus vizinhos.

Em t=33,0 o nodo 8 dissemina a mensagem para o nodo 7, nodo 4 e nodo 16. O nodo 2 dissemina a mensagem para o nodo 1 (redundante), nodo 4 (redundante) e nodo 10. O nodo 14 dissemina a mensagem para o nodo 13 (redundante), nodo 10 (redundante) e nodo 16 (redundante). O nodo 3 dissemina a mensagem para o nodo 4 (redundante), nodo 7 (redundante) e nodo 11. O nodo 9 dissemina a mensagem para o nodo 13 (redundante), nodo 11 (redundante) e nodo 10 (redundante). O nodo 15 dissemina a mensagem para o nodo 11 (redundante), nodo 16 (redundante) e nodo 7 (redundante). Todos os nodos processam a mensagem e disseminam para os seus vizinhos.

Em t=34,0 o nodo 7 dissemina a mensagem para o nodo 3 (redundante) e nodo 15 (redundante). Ao tentar disseminar a mensagem para o nodo 5, o nodo 7 detecta a falha no enlace que o conecta ao nodo 5, monta a mensagem de disseminação e prepara-se para iniciar o processo de disseminação da mensagem em t=35,0. A mensagem de disseminação montada pelo nodo 7 é apresentada na figura 19B. O nodo 4 dissemina a mensagem para o nodo 2 (redundante), nodo 3 (redundante) e nodo 12. O nodo 16 dissemina a mensagem para o nodo 14 (redundante), nodo 15 (redundante) e nodo 12 (redundante). O nodo 10

dissemina a mensagem para o nodo 9 (redundante), nodo 14 (redundante) e nodo 12 (redundante). O nodo 11 dissemina a mensagem para o nodo 9 (redundante), nodo 15 (redundante) e nodo 12 (redundante). Todos os nodos processam a mensagem e disseminam para os seus vizinhos. Em t=35,0 o nodo 12 dissemina a mensagem para o nodo 10 (redundante), nodo 16 (redundante) e nodo 11 (redundante). Neste momento o nodo 7 inicia a disseminação da nova mensagem para os seus vizinhos: o nodo 8, nodo 3 e nodo 15. O nodo 8, nodo 3 e nodo 15 processam-na e disseminam para os seus vizinhos. O processo se repete até que todos os nodos tenham processado e disseminado a mensagem em t=39,0.

Em resumo os resultados são apresentados na tabela abaixo:

Total de mensagens de diagnóstico	**Mensagens redundantes**	**Tempo total para o diagnóstico (em unidades de tempo)**
94	64	9

Neste exemplo o número total de mensagens de diagnóstico na rede foi 94, sendo 64 mensagens redundantes. O tempo total para completar o diagnóstico foi de 9 unidades de tempo. Mais uma vez observamos uma grande porcentagem de mensagens redundantes: 70% do total.

4.4 Uma Topologia Randômica

Este experimento foi realizado através da criação de um programa para gerar uma topologia randômica. Padronizou-se o número de vértices igual a 50 e a probabilidade de haver uma aresta entre um par de vértices de 10%. Desta forma, para cada par de vértices possível um número aleatório entre 1 e 10 é gerado. Se o número gerado for igual a 1 existe uma aresta entre o par de vértices; caso contrário não há aresta entre eles.

A tabela abaixo contém a configuração de uma topologia simulada, no qual a execução do algoritmo é descrita a seguir. Deve-se observar que o grafo é denso, com grau médio igual a 6.

Do vértice	Para os vértices
1	36 26 25 18 17 2
2	18 17 15 13 1
3	38 14 4
4	50 44 43 20 17 3
5	37 34 32 11
6	49 48 35 23
7	31 26
8	46 41 33 31 11
9	49 48 24
10	44 33 21 20 19 12
11	36 33 29 18 15 13 8 5
12	29 23 15 10
13	48 25 22 19 11 2
14	48 47 46 45 3
15	48 42 39 29 27 19 12 11 2
16	46 35 34 31 25
17	22 4 2 1
18	50 37 33 22 11 2 1
19	43 37 29 23 22 15 13 10
20	50 47 45 39 38 35 10 4
21	30 25 10
22	19 18 17 13
23	42 38 34 32 19 12 6
24	31 9
25	31 28 21 16 13 1
26	49 40 36 7 1
27	44 40 30 15
28	50 40 38 33 31 25
29	50 37 19 15 12 11
30	27 21
31	48 45 34 28 25 24 16 8 7
32	40 23 5
33	38 28 18 11 10 8
34	39 31 23 16 5
35	48 46 40 39 20 16 6
36	48 26 11 1
37	48 40 29 19 18 5
38	45 33 28 23 20 3
39	46 42 35 34 20 15
40	37 35 32 28 27 26
41	48 46 8
42	49 48 39 23 15
43	19 4
44	27 10 4

Do vértice	Para os vértices
45	38 31 20 14
46	41 39 35 16 14 8
47	20 14
48	50 49 42 41 37 36 35 31 15 14 13 9 6
49	48 42 26 9 6
50	48 29 28 20 18 4

O algoritmo inicia sua execução em t=0,0, o enlace entre o nodo 2 e o nodo 18 torna-se falho no tempo t=20,0. Na primeira rodada de testes todos os nodos testam o enlace para seus vizinhos. Assim o nodo 1 testa o enlace para o nodo 36, nodo 26, nodo 25, nodo 18, nodo 17 e nodo 2; o nodo 2 testa o enlace para o nodo 18, nodo 17, nodo 15, nodo 13 e nodo 1 e assim sucessivamente. Neste intervalo de testes os nodos testadores não detectam um novo evento e os testes são reiniciados em t=30,0. No instante t=20,0 o enlace entre o nodo 2 e o nodo 18 tornar-se-á falho. Em t=30,0, o nodo 2 ao testar o enlace para o nodo 18, nodo 17, nodo 15, nodo 13 e nodo 1 detecta a falha no enlace que o conecta ao nodo 18. Neste momento o nodo 2 prepara-se para iniciar a disseminação da mensagem que será enviada aos nodos da rede. A mensagem de disseminação montada pelo nodo 2 é mostrada na figura 20A.

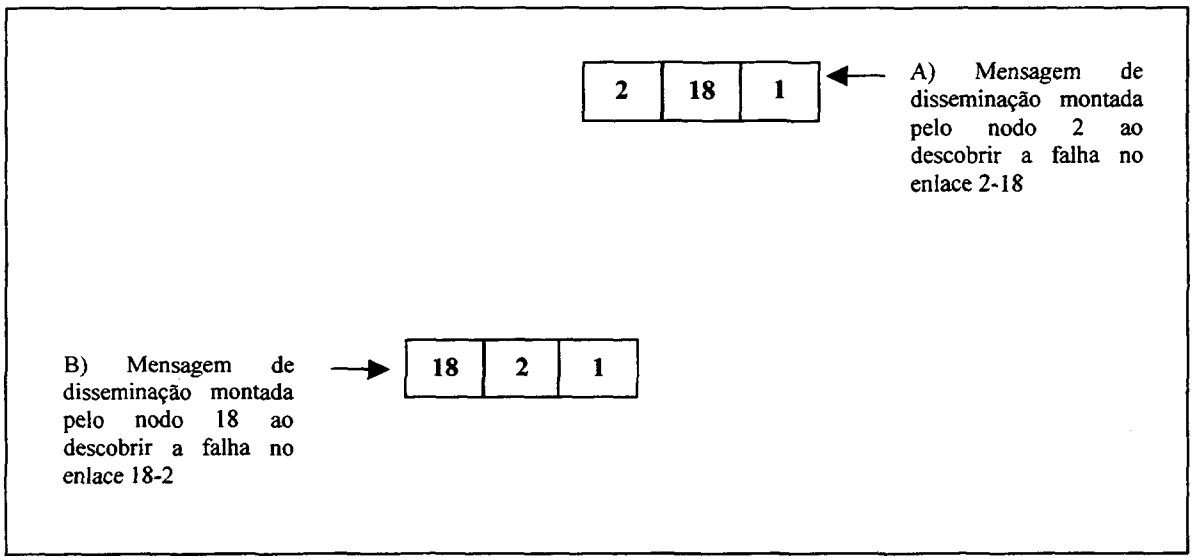


Figura 20: Mensagens de disseminação montada pelo nodo 2 e nodo 18.

Em t=31,0 o nodo 2 dissemina a mensagem para os seus vizinhos: o nodo 17, nodo 15, nodo 13 e nodo 1. Todos os nodos ao receberem a mensagem comparam as suas informações com as informações contidas na mensagem, como a informação não é conhecida eles atualizam as suas informações locais e disseminam a mensagem para os seus vizinhos. Em t=32,0 o nodo 17 dissemina a mensagem para o nodo 22, nodo 4 e nodo 1 (redundante). O nodo 15 dissemina a mensagem para o nodo 48, nodo 42, nodo 39, nodo 29, nodo 27, nodo 19, nodo 12 e nodo 11. O nodo 13 dissemina a mensagem para o nodo 48 (redundante), nodo 25, nodo 22 (redundante), nodo 19 (redundante) e nodo 11 (redundante). O nodo 1 dissemina a mensagem para o nodo 36, nodo 26, nodo 25 (redundante), nodo 18 e nodo 17 (redundante). Os nodos que recebem a mensagem processam-na e disseminam para os seus vizinhos.

Em t=33,0 o nodo 22 dissemina a mensagem para o nodo 19 (redundante), nodo 18 (redundante) e nodo 13 (redundante). O nodo 4 dissemina a mensagem para o nodo 50, nodo 44, nodo 43, nodo 20 e nodo 3. O nodo 48 dissemina a mensagem para o nodo 50 (redundante), nodo 49, nodo 42 (redundante), nodo 41, nodo 37, nodo 36 (redundante), nodo 35, nodo 31, nodo 14, nodo 13 (redundante), nodo 9 e nodo 6. O nodo 42 dissemina a mensagem para o nodo 49 (redundante), nodo 48 (redundante), nodo 39 (redundante) e nodo 23. O nodo 39 dissemina a mensagem para o nodo 46, nodo 42 (redundante), nodo 35 (redundante), nodo 34 e nodo 20 (redundante). O nodo 29 dissemina a mensagem para o nodo 50 (redundante), nodo 37 (redundante), nodo 19 (redundante), nodo 12 (redundante) e nodo 11 (redundante). O nodo 27 dissemina a mensagem para o nodo 44 (redundante), nodo 40 e nodo 30. O nodo 19 dissemina a mensagem para o nodo 43 (redundante), nodo 37 (redundante), nodo 29 (redundante), nodo 23 (redundante), nodo 22 (redundante), nodo 13 (redundante) e nodo 10. O nodo 12 dissemina a mensagem para o nodo 29 (redundante), nodo 23 (redundante) e nodo 10 (redundante). O nodo 11 dissemina

a mensagem para o nodo 36 (redundante), nodo 33, nodo 29 (redundante), nodo 18 (redundante), nodo 13 (redundante), nodo 8 e nodo 5. O nodo 25 dissemina a mensagem para o nodo 31 (redundante), nodo 28, nodo 21, nodo 16 e nodo 1 (redundante). O nodo 36 dissemina a mensagem para o nodo 48 (redundante), nodo 26 (redundante) e nodo 11 (redundante). O nodo 26 dissemina a mensagem para o nodo 49 (redundante), nodo 40 (redundante), nodo 36 (redundante) e nodo 7. O nodo 18 dissemina a mensagem para o nodo 50 (redundante), nodo 37 (redundante), nodo 33 (redundante), nodo 22 (redundante) e nodo 11 (redundante). O nodo 18 ao tentar disseminar a mensagem para o nodo 2 detecta a falha no enlace que o conecta ao nodo 2. O nodo 18 monta a mensagem de disseminação como mostra a figura 20B e prepara-se para iniciar o processo de disseminação da mensagem em t=34,0. Os nodos que recebem a mensagem processam-na e disseminam para os seus vizinhos.

Em t=34,0 o nodo 50 dissemina a mensagem para o nodo 48 (redundante), nodo 29 (redundante), nodo 28 (redundante), nodo 20 (redundante) e nodo 18 (redundante). O nodo 44 dissemina a mensagem para o nodo 27 (redundante) e nodo 10 (redundante). O nodo 43 dissemina a mensagem para o nodo 19 (redundante). O nodo 20 dissemina a mensagem para o nodo 50 (redundante), nodo 47, nodo 45, nodo 39 (redundante), nodo 38, nodo 35 (redundante) e nodo 10. O nodo 3 dissemina a mensagem para o nodo 38 (redundante) e nodo 14 (redundante). Ainda em t=34,0 o nodo 49, nodo 41, nodo 37, nodo 35, nodo 31, nodo 14, nodo 9, nodo 6, nodo 23, nodo 46, nodo 34, nodo 40, nodo 30, nodo 10, nodo 33, nodo 8, nodo 5, nodo 28, nodo 21, nodo 16 e nodo 7 disseminam a mensagem. O nodo 18 dissemina a nova mensagem para os seus vizinhos: o nodo 50, nodo 37, nodo 33, nodo 22, nodo 11 e nodo 1. Os nodos que recebem a mensagem processam-na e disseminam para os seus vizinhos. O processo se repete até t=38,0 quando todos os nodos processaram e disseminaram a mensagem.

Em resumo os resultados são apresentados na tabela abaixo:

Total de mensagens de diagnóstico	Mensagens redundantes	Tempo total para o diagnóstico (em unidades de tempo)
418	320	8

Neste exemplo o número total de mensagens de diagnóstico na rede foi 418, sendo 320 mensagens redundantes, aproximadamente 76% do total. O tempo total para completar o diagnóstico foi de 8 unidades de tempo.

4.5 Topologia da Rede Nacional de Pesquisa (RNP)

A topologia da RNP [20] é apresentada na figura 21. A tabela a seguir contém a numeração dos nodos correspondentes a cada cidade brasileira de acordo com a simulação realizada.

Número do nodo	Cidade brasileira correspondente
1	Brasília
2	Goiânia
3	Campo Grande
4	Rio Branco
5	Porto Velho
6	Cuiabá
7	Manaus
8	Palmas
9	Boa Vista
10	Macapá
11	Belém
12	E.U.A.
13	Rio de Janeiro
14	São Paulo

Número do nodo	Cidade brasileira correspondente
15	Belo Horizonte
16	Vitória
17	Fortaleza
18	Salvador
19	Recife
20	Porto Alegre
21	Florianópolis
22	Curitiba
23	São Luís
24	Teresina
25	Maceió
26	Aracaju
27	Natal
28	Campina Grande

O algoritmo inicia sua execução em t=0,0. O enlace entre o nodo 1 (Brasília) e o nodo 14 (São Paulo) torna-se falho no tempo t=20,0. Na primeira rodada de testes todos os nodos testam o enlace para seus vizinhos. Assim o nodo 1 testa o enlace para o nodo 2, nodo 3, nodo 4, nodo 5, nodo 6, nodo 7, nodo 8, nodo 9, nodo 10, nodo 11, nodo 12, nodo 13 e nodo 14; o nodo 2 testa o enlace para o nodo 1; o nodo 3 testa o enlace para o nodo 1 e assim sucessivamente. Neste intervalo de testes os nodos testadores não detectam um novo evento e os testes são reiniciados em t=30,0. No instante t=20,0 o enlace entre o nodo 1 e o nodo 14 tornar-se-á falho. O nodo 1 ao testar o enlace para o nodo 2, nodo 3, nodo 4 até o nodo 14 detecta a falha no enlace que o conecta ao nodo 14. Neste momento o nodo 1 prepara-se para iniciar a disseminação da mensagem que será enviada aos nodos da rede. A mensagem de disseminação montada pelo nodo 1 é mostrada na figura 22A.

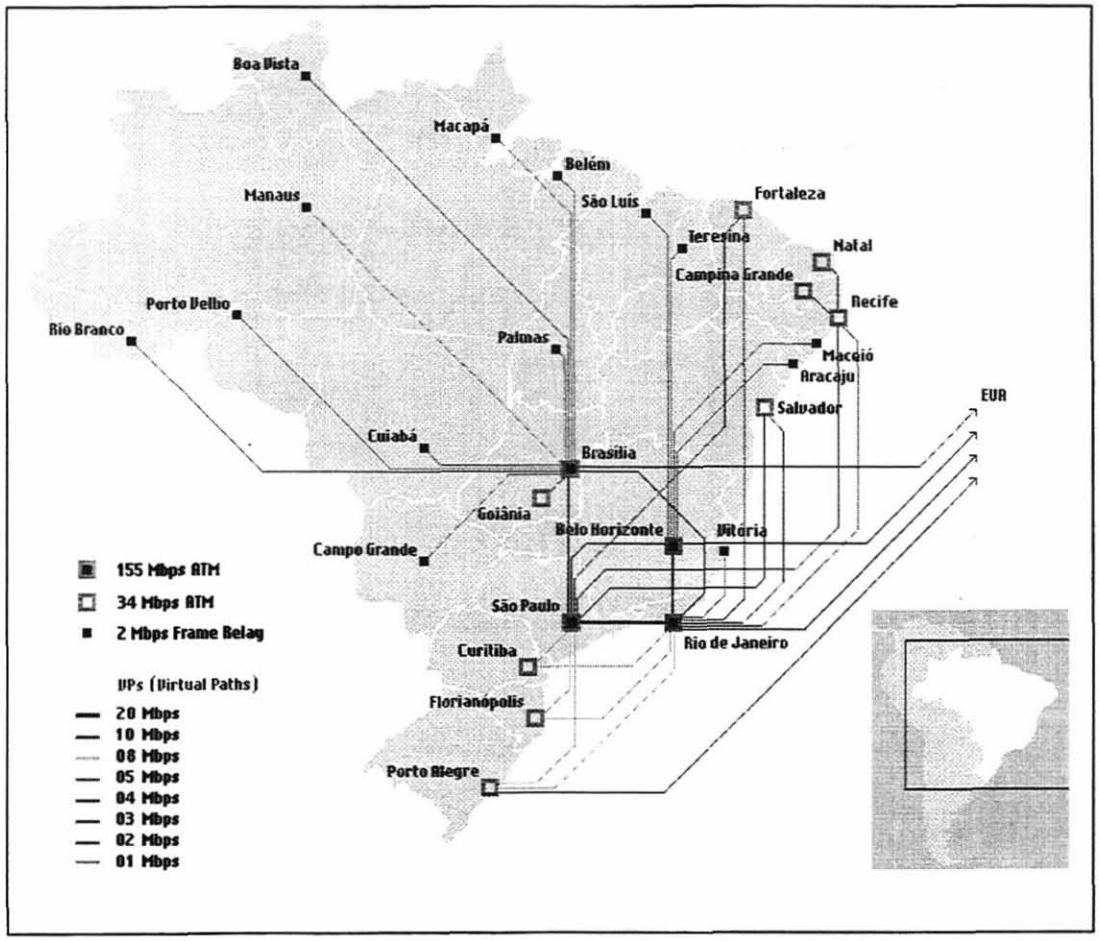


Figura 21: Topologia da RNP.

Em t=31,0 o nodo 1 dissemina a mensagem para os seus vizinhos: o nodo 2, nodo 3, nodo 4, nodo 5, nodo 6, nodo 7, nodo 8, nodo 9, nodo 10, nodo 11, nodo 12 e nodo 13. Todos os nodos ao receberem a mensagem comparam as suas informações com as informações contidas na mensagem, como a informação não é conhecida eles atualizam as suas informações locais e disseminam a mensagem para os seus vizinhos. Em t=32,0 o nodo 12 dissemina a mensagem para o nodo 15, nodo 13 (redundante) e nodo 20. O nodo 13 dissemina a mensagem para o nodo 20 (redundante), nodo 21, nodo 22, nodo 14, nodo 16, nodo 17, nodo 18, nodo 19, nodo 12 (redundante) e nodo 15 (redundante). Os nodos que recebem a mensagem processam-na e disseminam para os seus vizinhos.

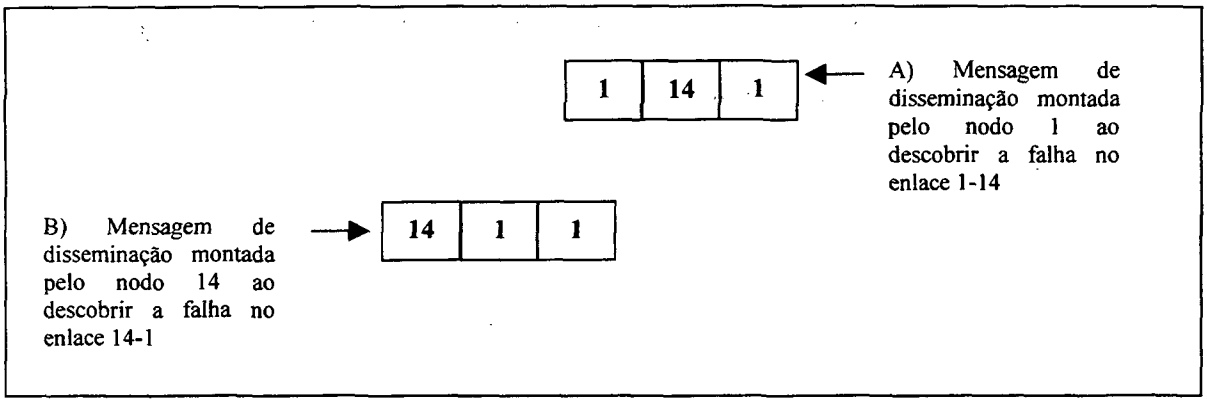


Figura 22: Mensagens de disseminação montada pelo nodo 1 e nodo 14.

Em t=33,0 o nodo 15 dissemina a mensagem para o nodo 14 (redundante), nodo 13 (redundante), nodo 23, nodo 24, nodo 25 e nodo 26. O nodo 20 dissemina a mensagem para o nodo 13 (redundante) e nodo 14 (redundante). O nodo 21 dissemina a mensagem para o nodo 14 (redundante). O nodo 22 dissemina a mensagem para o nodo 14 (redundante). O nodo 14 dissemina a mensagem para o nodo 22 (redundante), nodo 21 (redundante), nodo 20 (redundante), nodo 15 (redundante), nodo 17 (redundante), nodo 19 (redundante) e nodo 18 (redundante). O nodo 14 ao tentar disseminar a mensagem para o nodo 1 detecta a falha no enlace que o conecta ao nodo 1. O nodo 14 monta a mensagem

de disseminação como mostra a figura 22B, e prepara-se para iniciar o processo de disseminação da mensagem em t=34,0. O nodo 17 dissemina a mensagem para o nodo 14 (redundante). O nodo 18 dissemina a mensagem para o nodo 14 (redundante). O nodo 19 dissemina a mensagem para o nodo 14 (redundante), nodo 27 e nodo 28. Os nodos que recebem a mensagem processam-na e disseminam para os seus vizinhos.

Em t=34,0 o nodo 14 dissemina a nova mensagem para os seus vizinhos: o nodo 22, nodo 21, nodo 20, nodo 19, nodo 18, nodo 17, nodo 15 e nodo 13. O processo se repete até t=36,0 quando todos os nodos processaram e disseminaram a mensagem.

Em resumo os resultados são apresentados na tabela abaixo:

Total de mensagens de diagnóstico	**Mensagens redundantes**	**Tempo total para o diagnóstico (em unidades de tempo)**
94	40	6

Neste exemplo o número total de mensagens de diagnóstico na rede foi 94, sendo 40 mensagens redundantes. O tempo total para completar o diagnóstico foi de 6 unidades de tempo.

4.6 Comparações

Esta seção contém resultados de simulação do algoritmo em comparação ao algoritmo do Agente Chinês e ao algoritmo NBND. Os parâmetros considerados são o total de mensagens de diagnóstico, o número de mensagens redundantes e o tempo total para realizar o diagnóstico.

4.6.1 Uma Topologia Exemplo

A tabela a seguir contém resultados comparativos da execução dos algoritmos com a configuração do grafo da topologia exemplo descrita na seção 4.1.

Algoritmo	Total de mensagens de diagnóstico	Mensagens redundantes	Tempo para realizar o diagnóstico (em unidades de tempo)
O algoritmo proposto	28	16	7
O algoritmo do Agente Chinês	7	1	7
O algoritmo NBND	12	0	9

Na simulação realizada com o grafo da topologia exemplo os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 28 mensagens, das quais 16 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 7 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês são o total de mensagens de diagnóstico igual a 7 mensagens, das quais 1 mensagem é redundante. O tempo total para realizar o diagnóstico é de 7 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 12 mensagens. O tempo total para realizar o diagnóstico é de 9 unidades de tempo.

Esta foi a única topologia na qual o algoritmo proposto apresentou a mesma latência do algoritmo do Agente Chinês, que em geral é bem mais lento. Uma discussão do impacto das mensagens redundantes segue na seção 4.7.

4.6.2 Topologia $D_{1,2}$

A tabela a seguir contém resultados comparativos da execução dos algoritmos com a configuração do grafo da topologia $D_{1,2}$ com 9 vértices descrita na seção 4.2.

Algoritmo	Total de mensagens de diagnóstico	Mensagens redundantes	Tempo para realizar o diagnóstico (em unidades de tempo)
O algoritmo proposto	52	36	7
O algoritmo do Agente Chinês	13	5	13
O algoritmo NBND	16	0	9

Na simulação realizada com o grafo da topologia $D_{1,2}$ os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 52 mensagens, das quais 36 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 7 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês são o total de mensagens de diagnóstico igual a 13 mensagens, das quais 5 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 13 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 16 mensagens. O tempo total para realizar o diagnóstico é de 9 unidades de tempo.

Neste caso o algoritmo proposto apresentou a melhor latência. O algoritmo do Agente Chinês, que apresenta o menor impacto no desempenho da rede, resultou em uma latência duas vezes maior que a do algoritmo proposto. O algoritmo NBND que não utiliza mensagens redundantes também apresentou latência maior que a do algoritmo proposto. Uma discussão do impacto das mensagens redundantes segue na seção 4.7.

4.6.3 Topologia Hipercubo

A tabela a seguir contém resultados comparativos da execução dos algoritmos com a configuração do grafo da topologia hipercubo com 16 nodos descrita na seção 4.3.

Algoritmo	Total de mensagens de diagnóstico	Mensagens redundantes	Tempo para realizar o diagnóstico (em unidades de tempo)
O algoritmo proposto	94	64	9
O algoritmo do Agente Chinês	28	14	28
O algoritmo NBND	30	0	12

Na simulação realizada com o hipercubo de 16 nodos os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 94 mensagens, das quais 64 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 9 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês indicam um total de mensagens de diagnóstico igual a 28 mensagens, das quais 14 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 28 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 30 mensagens. O tempo total para realizar o diagnóstico é de 12 unidades de tempo.

Para um hipercubo com 64 nodos os resultados são apresentados na tabela abaixo.

Algoritmo	Total de mensagens de diagnóstico	Mensagens redundantes	Tempo para realizar o diagnóstico (em unidades de tempo)
O algoritmo proposto	478	352	15
O algoritmo do Agente Chinês	156	93	156
O algoritmo NBND	126	0	24

Na simulação realizada com o hipercubo de 64 nodos os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 478 mensagens, das quais 352 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 15 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês são o total de mensagens de diagnóstico igual a 156 mensagens, das quais 93 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 156 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 126 mensagens. O tempo total para realizar o diagnóstico é de 24 unidades de tempo.

Para um hipercubo com 128 nodos os resultados são apresentados na tabela abaixo.

Algoritmo	Total de mensagens de diagnóstico	Mensagens redundantes	Tempo para realizar o diagnóstico (em unidades de tempo)
O algoritmo proposto	990	736	23
O algoritmo do Agente Chinês	348	221	348
O algoritmo NBND	254	0	36

Na simulação realizada com o hipercubo de 128 nodos os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 990 mensagens, das quais 736 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 23 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês são o total de mensagens de diagnóstico igual a 348 mensagens, das quais 221 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 348 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 254 mensagens. O tempo total para realizar o diagnóstico é de 36 unidades de tempo.

Tanto nos hipercubos de 16 e 64 nodos quanto no hipercubo de 128 nodos, o algoritmo proposto apresentou a melhor latência possível. Uma discussão do impacto das mensagens redundantes segue na seção 4.7.

4.6.4 Uma Topologia Randômica

A tabela a seguir contém resultados comparativos da execução dos algoritmos com a configuração do grafo da topologia randômica descrita na seção 4.4.

Algoritmo	**Total de mensagens de diagnóstico**	**Mensagens redundantes**	**Tempo para realizar o diagnóstico (em unidades de tempo)**
O algoritmo proposto	418	320	8
O algoritmo do Agente Chinês	109	60	109
O algoritmo NBND	143	0	14

Na simulação realizada com o grafo da topologia randômica os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 418

mensagens, das quais 320 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 8 unidades de tempo. Simulações em 30 topologias randômicas foram realizadas. Os resultados médios obtidos foram: média das mensagens de diagnóstico igual a 393 mensagens, média das mensagens redundantes igual a 295 mensagens e o tempo médio para realizar o diagnóstico de 9 unidades de tempo. O desvio padrão obtido foi igual a 42.92, para as mensagens de diagnóstico e mensagens redundantes e igual a 0.8 para o tempo de diagnóstico.

Os resultados obtidos pelo algoritmo do Agente Chinês, na simulação de uma topologia randômica, são o total de mensagens de diagnóstico igual a 109 mensagens, das quais 60 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 109 unidades de tempo.

No algoritmo NBND os resultados obtidos, na simulação de uma topologia randômica, são o total de mensagens de diagnóstico igual a 143 mensagens. O tempo total para realizar o diagnóstico é de 14 unidades de tempo.

Mais uma vez o algoritmo proposto apresentou a melhor latência possível. Uma discussão do impacto das mensagens redundantes segue na seção 4.7.

4.6.5 Topologia da Rede Nacional de Pesquisa (RNP)

A tabela a seguir contém resultados comparativos da execução dos algoritmos com a configuração da topologia RNP descrita na seção 4.5.

Algoritmo	**Total de mensagens de diagnóstico**	**Mensagens redundantes**	**Tempo para realizar o diagnóstico (em unidades de tempo)**
O algoritmo proposto	94	40	6
O algoritmo do Agente Chinês	55	27	55
O algoritmo NBND	54	0	9

Na simulação realizada com a RNP os resultados obtidos pelo algoritmo proposto são o total de mensagens de diagnóstico igual a 94 mensagens, das quais 40 mensagens são redundantes. O tempo total para realizar o diagnóstico é de 6 unidades de tempo.

Os resultados obtidos pelo algoritmo do Agente Chinês são o total de mensagens de diagnóstico igual a 55 mensagens, das quais 27 mensagem são redundantes. O tempo total para realizar o diagnóstico é de 55 unidades de tempo.

No algoritmo NBND os resultados obtidos são o total de mensagens de diagnóstico igual a 54 mensagens. O tempo total para realizar o diagnóstico é de 9 unidades de tempo.

Mais uma vez o algoritmo proposto apresentou a melhor latência possível. Uma discussão do impacto das mensagens redundantes segue na seção 4.7.

4.7 Considerações Finais sobre as Comparações

Se um nodo recebe uma mensagem redundante, é porque existem dois caminhos disjuntos entre o nodo que gerou a mensagem e o nodo que a recebe. A figura 23 ilustra este fato. O nodo 1 gera uma mensagem de diagnóstico, que é recebida duas vezes pelo nodo 3, pois há dois caminhos disjuntos a serem percorridos entre o nodo 1 e o nodo 3: o caminho que usa o enlace 1-3 e o caminho que usa o enlace 1-2 e enlace 2-3. Desta forma, se durante a disseminação da mensagem de diagnóstico novos eventos de falha ocorrerem na rede, a redundância vai permitir que o algoritmo tolere falhas de caminhos, tantas quantas são as mensagens redundantes. Para o exemplo da figura 23, o nodo 3 recebe duas mensagens redundantes, portanto o algoritmo tolera a falha de um caminho, funcionando mesmo na presença de uma falha extra durante a disseminação.

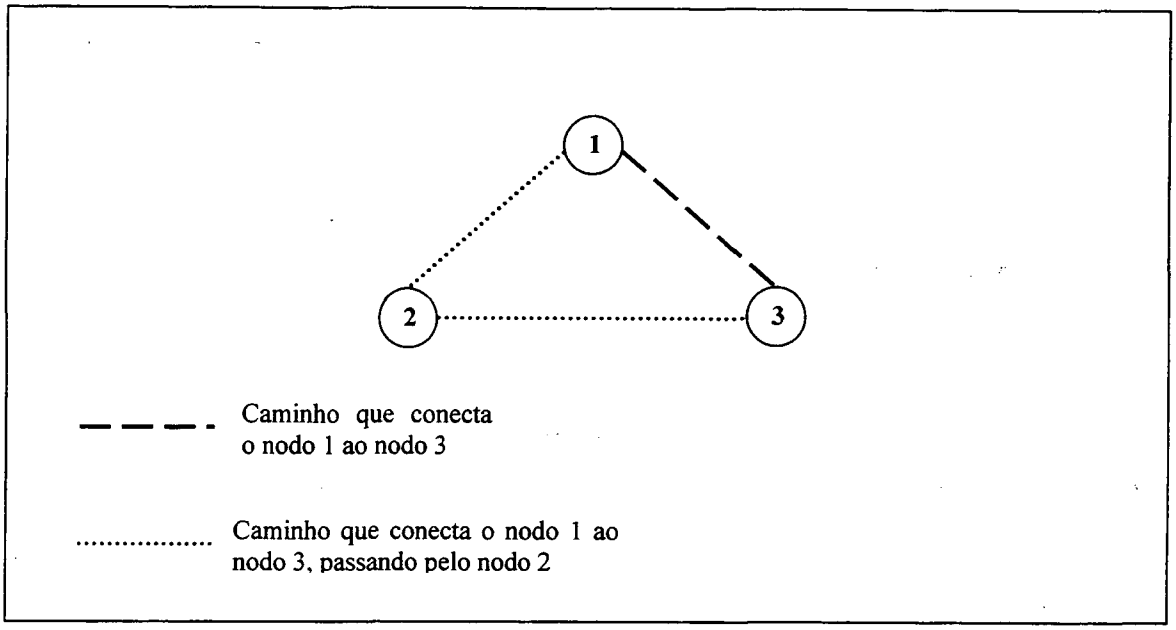


Figura 23: Exemplo de caminhos disjuntos.

Assim, o número de mensagens redundantes é, na verdade, considerado uma vantagem do algoritmo, quando comparado à outras abordagens. Os algoritmos de diagnóstico são justamente usados para permitir que os nodos sem falhas possam recuperar-se quando o sistema está parcialmente inoperante. Portanto é fundamental que os algoritmos sejam tolerantes a falhas. Por outro lado, as mensagens são pequenas, e o número máximo de mensagens por evento é 2*L, onde L é o número de enlaces no sistema. Desta forma o uso do algoritmo baseado em inundação é justificado, exatamente devido à redundância intrínseca e seu baixo impacto no desempenho do sistema.

O próximo capítulo apresenta as conclusões deste trabalho.

Capítulo 5

Conclusão

Apresentamos neste trabalho um novo algoritmo para diagnóstico de redes de topologia arbitrária utilizando a técnica de disseminação de mensagens conhecida como inundação ou *flooding*. Esta estratégia é bem mais simples do que as estratégias empregadas por outros algoritmos, como a árvore geradora mínima utilizada no algoritmo *Adapt*, árvore *breadth-first* utilizada no algoritmo NBND, ou o percurso do Agente Chinês.

O algoritmo apresentado detecta falhas nos enlaces de comunicação e ao descobrir um novo evento, envia paralelamente aos nodos alcançáveis da rede uma mensagem de disseminação avisando-os do evento ocorrido. O tamanho da mensagem de disseminação utilizada é pequeno, não apresentando impacto significativo no desempenho da rede. Um mecanismo de descarte de mensagens redundantes é utilizado, ou seja, um nodo ao receber uma mensagem de disseminação compara as informações recebidas com as suas informações locais de diagnóstico, garantindo, desta forma, que não processe mensagens redundantes.

De acordo com os resultados obtidos através de simulações, podemos constatar que a latência do algoritmo é proporcional ao diâmetro da rede. Confirmamos também que o número de mensagens redundantes geradas é menor que o máximo possível, ou seja, o dobro do número de enlaces.

Como trabalhos futuros deve-se estudar o comportamento do algoritmo na presença de falhas que particionam a rede. Um mecanismo para a recuperação de falhas e o procedimento para a junção de componentes conexos, cada um dos quais é uma rede de topologia arbitrária em si próprio, também deve ser especificado e analisado. A formalização da especificação do algoritmo também é meta de trabalhos futuros.

Referências Bibliográficas

[1] A. Bagchi, and S. L. Hakimi, "An Optimal Algorithm for Distributed System-Level Diagnosis," *Proc. 21st Fault Tolerant Computing Symp*, June, 1991.

[2] R. P. Bianchini, and R. Buskens, "No Adaptative Distributed System-Level Diagnosis Algorithm and Its Implementation," *Proc. 21st Fault Tolerant Computing Symp*, pp. 222-229, 1991.

[3] R. Bianchini, M. Stahl, and R. Buskens, "Simulation of the Adapt2 On-Line Diagnosis Algorithm for General Topology Networks," *Proc. IEEE 11th Symp. Reliable Distributed Systems*, October, 1992.

[4] R. Bianchini, M. Stahl, and R. Buskens, "The Adapt2 On-line Diagnosis Algorithm for General Topology Networks," *Proc. Globecom*, pp. 610-614, 1992.

[5] R. P. Bianchini, K. Goodwin, and D. S. Nydick, "Practical Application and Implementation of System-Level Diagnosis Theory," *Proc. 20th Fault Tolerant Computing Symp*, pp. 332-339, 1990.

[6] E. P. Duarte Jr., "Um Algoritmo para Diagnóstico de Redes de Topologia Arbitrária," *I Workshop de Tolerância a Falhas da SBC*, Porto Alegre, 1998.

[7] E. P. Duarte Jr., J. M. A. P. Cestari, "O Agente Chinês para Diagnóstico de Redes de Topologia Arbitrária", *II Workshop de Testes & Tolerância a Falhas da SBC*, pp. 88-93, Curitiba, 2000.

[8] E. P. Duarte Jr., G. O. Mattos, "Diagnóstico de Redes de Topologia Arbitrária: Um Algoritmo Baseado em Inundação de Mensagens", *II Workshop de Testes & Tolerância a Falhas da SBC*, pp. 82-87, Curitiba, 2000.

[9] E. P. Duarte Jr., and T. Nanya, "Hierarchical Distributed System-Level Diagnosis Applied for SNMP-based Network Fault Management," *Proc. IEEE 16th Symp. Reliable Distributed Systems*, Niagara, September, 1996.

[10] E. P. Duarte Jr., T. Nanya, G. Mansfield, and S. Nogushi, "Non-Broadcast Network Fault-Monitoring Based on System-Level Diagnosis," *Proc. IEEE/IFIP IM'97*, 1997.

[11] S. L. Hakimi, and A. T. Amin, "Characterization of Connection Assignment of Diagnosable Systems," *IEEE Trans. Comput.*, vol. C-23, Jan, 1974.

[12] S. L. Hakimi, and K. Nakajima, "On Adaptative System Diagnosis," *IEEE Transactions on Computers*, Vol. 33, pp. 234-240, 1984.

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

[14] J. G. Kuhl, and S. M. Reddy, "Distributed Fault-Tolerance for Large Multiprocessor Systems," *Proc. 7th Annual Symp. Computer Architecture*, pp. 23-30, 1980.

[15] J. G. Kuhl, and S. M. Reddy, "Fault-Diagnosis in Fully Distributed Systems," *Proc. 11th Fault Tolerant Computing Symp*, pp. 100-105, 1981.

[16] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*, The MIT Press, Cambridge, MA, 1987.

[17] G. Masson, D. Blough, and G. Sullivan, "System Diagnosis," in Fault-Tolerant Computer System Design, ed. D.K. Pradhan, Prentice-Hall, 1996.

[18] F. P. Preparata, G. Metze, and R. T. Chien, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Trans. Electrom. Comput.*, vol. EC-16, pp. 848-854, Dec, 1967.

[19] S. Rangarajan, A. T. Dahbura, and E. A. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Transactions and Computers*, Vol. 44, pp. 312-333, 1995.

[20] RNP – Rede Nacional de Pesquisa. http://www.rnp.br/backbone/bkb-mapa.html

[21] M.T. Rose, *The Simple Book - An Introduction to Internet Management*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1994.

[22] J. I. Siqueira, E. Fabris, E. P. Duarte Jr., "A Token Based Testing Strategy for Non-Broadcast Network Diagnosis," *1st IEEE Latin American Test Workshop*, pp. 166-171, Rio de Janeiro, 2000.

[23] R. J. Wilson, J. J. Watkins, *Graphs: An Introduction Approach*, John Wiley & Sons, New York, 1990.

[24] R. Sedgewick, *Algorithms in C*, Addison-Wesley, Reading, MA, 1992.

[25] A. S. Tanenbaum, *Computer Networks – 3rd Edition*, Prentice Hall, 1995.

Apêndice A

Programa de Simulação

```
/* ************************************************************************
 *      UNIVERSIDADE FEDERAL DO PARANA                                    *
 *      MESTRADO EM INFORMATICA                                           *
 *      ORIENTADOR: PROF. DR. ELIAS P. DUARTE JR.                         *
 *                                                                        *
 *      TRABALHO DE DISSERTACAO DE MESTRADO                               *
 *      DIAGNOSTICO DE REDES DE TOPOLOGIA ARBITRARIA                      *
 *                                                                        *
 *      DESENVOLVIDO POR GIORGIA DE OLIVEIRA MATTOS - FEVEREIRO/2001      *
 *                                                                        *
************************************************************************ */

#include <stdlib.h>
#include <stdio.h>
#include "smpl.h"


#define teste 1         /* EVENTOS */
#define dissemina 2
#define falha 3

#define TRUE 1
#define FALSE 0
#define maxV 2000


struct node      /* IDENTIFICADOR DO NODO */
{
   int v,       /* Numero do vertice */
       cont;    /* Contador de estados, indica se o nodo esta ativo */
   struct node *next;    /* Ponteiro para o proximo nodo */
};


typedef struct    /* DESCRITOR DO NODO */
{
   int id;                 /* SMPL facility identifier */
   struct node adj_node[maxV];  /* Copia da adlist para todos os nodos */
}tnodo;
```

```
struct msg
{
   int testador,        /* Numero do nodo testador      */
       testado,         /* Numero do nodo testado       */
       contador;        /* Contador de estados          */
};

int V,                  /* Numero de vertices           */
    E,                  /* Numero de arestas            */
    x, y,               /* Armazena o par de arestas */
    v_ini,              /* Contem o vertice inicial = 1 */
    num_msg = 0,        /* Total de mensagens gerada durante a simulacao */
    redundante = 0,     /* Total de msgs redundantes geradas na simulacao */
    link_status_counter = 0; /* Contador de estados */

FILE *out;                      /* Arquivo de saida */
static char nomearq[20];        /* Nome do arquivo  */
struct node *z, *t;             /* Apontadores para os nodos */
struct node adj[maxV];          /* Contem a lista de adjacencias */
tnodo nodo[maxV];               /* Vetor que contem os nodos e suas
                                   listas de adjacencia */
struct msg msg_dissemina;    /* Contem a mensagem a ser disseminada */

int link_falho[2],              /* Contem o numero dos dois vertices que
                                   formam o link falho */
    link_falho2[2];             /* Contem o numero dos dois vertices que
                                   formam o link falho */
int link_sendo_testado[2],      /* Contem o numero dos dois vertices que
                                   formam o link sendo testado */
    link_sendo_testado2[2];



/* ************************************************************************
 *                                                                        *
 *                              F U N C O E S                             *
 *                                                                        *
************************************************************************ */
/* Constroi a lista de adjacencias */
adjlist ()
{
   int i, j;            /* Indices auxiliares   */
   struct node *novo;   /* Nodo auxiliar        */

   printf("\n\t\t Entrada com os dados do Grafo \n\n");
   printf("Numero de Vertices =  ");
   scanf("%d",&V);
   printf("Numero de Arestas  =  ");
   scanf("%d",&E);      /* Obtem numero de vertices e arestas */
   printf("\n");
   z = (struct node *) malloc (sizeof *z);
   z->next = z;
   for (j=1; j<=V; j++)
   {
      adj[j].next = z;
      for (i=1; i<=V; i++)
         nodo[i].adj_node[j].next = z;
   }
```

```
   for (j=1; j<=E; j++)
   {
      printf("\nVertice 1 = ");
      scanf("%d",&x);
      printf("Vertice 2 = ");
      scanf("%d",&y);   /* Obtem a sequencia de arestas */
      t = (struct node *) malloc (sizeof *t);
      t->v = x;
      t->cont = 0;
      t->next = adj[y].next;
      adj[y].next = t;

      for (i=1; i<=V; i++)  /* Salva a primeira aresta na lista de adj.*/
      {
         novo = (struct node *) malloc (sizeof *novo);
         novo->v = x;
         novo->cont = 0;
         novo->next = nodo[i].adj_node[y].next;
         nodo[i].adj_node[y].next = novo;
      }

      t = (struct node *) malloc (sizeof *t);
      t->v = y;
      t->cont = 0;
      t->next = adj[x].next;
      adj[x].next = t;
      for (i=1; i<=V; i++)  /* Salva a segunda aresta na lista de adj.*/
      {
         novo = (struct node *) malloc (sizeof *novo);
         novo->v = y;
         novo->cont = 0;
         novo->next = nodo[i].adj_node[x].next;
         nodo[i].adj_node[x].next = novo;
      }
   }
   v_ini = 1;          /* Obtem o vertice inicial */
   printf("\n\nVertice Inicial =  %d",v_ini);
   printf("\n\nDados do grafo\n");
   fprintf(out,"Dados do grafo\n");
   fprintf(out,"Numero de Vertices =  %d",V);
   fprintf(out,"\nNumero de Arestas  =  %d",E);
   for (j=1; j<=V; j++)
   {
      nodo[j].id = j;
      printf("\nDo vertice: %d p/ as arestas:  ",j);
      fprintf(out,"\nDo vertice: %d p/ as arestas:  ",j);
      for (t=adj[j].next; t!=z; t=t->next)
      {
         fprintf(out," %d ",t->v);
         printf(" %d  ",t->v);
      }
    }
    printf("\nVertice Inicial:  %d\n\n",v_ini);
    fprintf(out,"\nVertice Inicial:  %d\n\n",v_ini);
}


/* Avisa que houve um novo evento */
void grava(int testador, int testado, real tempo)
{
   printf("\nNodo %d obteve nova informacao do nodo %d, no tempo
```

```
            %4.1f\n",testador,testado,tempo);
    fprintf(out,"\nNodo %d obteve nova informacao do nodo %d, no tempo
                %4.1f\n",testador,testado,tempo);
}


/* Testa se um nodo esta ou nao falho */
int testa_nodo(int testador, int testado, real tempo)
{
   struct node *p;

   printf("\nNodo %d testa o nodo %d no tempo
            %5.1f",testador,testado,tempo);
   fprintf(out,"\nNodo %d testa o nodo %d no tempo
               %5.1f",testador,testado,tempo);
   p = nodo[testador].adj_node[testador].next;
   while (p->v != testado)
      p = p->next;

   /* par = sem falha; impar = com falha */
   link_status_counter = p->cont % 2;

   /* Verifica se eh um novo evento */
   if (status(testador) != link_status_counter)
   {
      grava(testador,testado,tempo);
      link_falho[0] = testador;
      link_falho[1] = testado;
      msg_dissemina.testador = link_falho[0];
      msg_dissemina.testado = link_falho[1];
      msg_dissemina.contador = link_status_counter;
      return (dissemina);       /* Eh novo evento */
   }
   else
      return (teste);           /* Nao eh novo evento */
}


/* Dissemina pela rede uma mensagem avisando a todos que ha um nodo falho
*/
dissemina_mensagem (int testador, int testado)
{
   struct node *p;

   /* Atualiza o campo cont do link falho na adjlist */
   p = nodo[testador].adj_node[testador].next;
   while ((p != z) && (p->v != testado))
      p = p->next;
   if (p->cont == msg_dissemina.contador)
      p->cont ++;
}


/* Verifica se todos os nodos receberam a mensagem dissemina */
int todos_disseminaram_msg (int dissem[])
{
   int i, end = FALSE;
```

```
   for (i=1; ((i<=V) && !end); i++)
      if (dissem[i] == 0)
         end = TRUE;
   return (end);
}


/* ********************************************************************
 *                                                                    *
 *                    F U N C A O     P R I N C I P A L               *
 *                                                                    *
 * ****************************************************************** */

main ()
{
    static int evento,       /* identificador do evento               */
               novo_evento,  /* indica novo evento                    */
               token,        /* identificador do nodo                 */
               token2,
               dissemina2=FALSE, /* indica a 2a. parte da disseminacao */
               acabou=FALSE,
               sair=TRUE,  /* indica se todos os nodos receberam a msg */
               sair2=TRUE;

    real relogio=0.0,     /* tempo corrente da simulacao    */
         relogio2=0.0,    /* tempo corrente da segunda disseminacao */
         relogiof=20.0;   /* tempo que o nodo torna-se falho        */

    struct node *aux, *aux2;           /* ponteiros auxiliares     */

    int nodo1f,               /* primeiro vertice do enlace falho      */
        nodo2f,               /* segundo vertice do enlace falho       */
        a, b, aa, bb;           /* indices auxiliares para os vetores  */

    int disseminaram [maxV], /* nodos que ja disseminaram a msg */
        disseminaram2 [maxV],
        de_quem_recebeu [maxV], /* indica quem enviou a msg */
        de_quem_recebeu2 [maxV],
        receberam [maxV],  /* indica os nodos que ja receberam a msg */
        receberam2 [maxV],
        disseminam_mesmo_tempo [maxV], /* nodos que disseminam a msg ao
                                          mesmo tempo */
        disseminam_mesmo_tempo2 [maxV];



    sprintf(nomearq,"%s","relat");
    if ((out = fopen(nomearq,"w")) == NULL)
    {
       fprintf(out,"ERRO - Arquivo de resultados nao foi aberto");
       printf("ERRO - Arquivo de resultados nao foi aberto\n");
       fflush(out);
       exit(-1);
    }

    smpl(0,"DRTA Simulation - Diag. de Redes de Topologia Arbitraria");
   /* Obtem dados do grafo e monta a lista de adjacencias */
   adjlist();
```

```
/* Obtem os dados do enlace falho */
printf("\nDados do enlace falho");
fprintf(out,"\nDados do enlace falho");
printf("\nVertice1: ");scanf("%d",&nodo1f);
printf("\nVertice2: ");scanf("%d",&nodo2f);
fprintf(out,"\nVertice1: %d",nodo1f);
fprintf(out,"\nVertice2: %d",nodo2f);

/* Inicializa os vetores */
for (a=1; a<=V; a++)
{
   disseminaram[a] = 0;
   disseminaram2[a] = 0;
   de_quem_recebeu[a] = 0;
   de_quem_recebeu2[a] = 0;
   receberam[a] = 0;
   receberam2[a] = 0;
   disseminam_mesmo_tempo[a] = 0;
   disseminam_mesmo_tempo[a] = 0;
}
printf("\n\nINICIANDO A SIMULACAO ......\n");
fprintf(out,"\n\nINICIANDO A SIMULACAO ......\n");
printf("\n\n\tE V E N T O   D E   T E S T E S");
fprintf(out,"\n\n\tE V E N T O   D E   T E S T E S");

/* Schedule TESTE no vertice inicial */
schedule(teste,relogio,v_ini);

novo_evento = teste;
link_falho[0] = link_falho[1] = 0;
link_falho2[0] = link_falho2[1] = 0;
a = aa = 1;   b = bb = 0;
msg_dissemina.testador = msg_dissemina.testado =
msg_dissemina.contador=0;

while ((relogio < 300.0) && (sair2))
{
   cause(&evento,&token);
   switch(evento)
   {
       case teste:
       {
          aux = nodo[token].adj_node[token].next;
          while ((aux != z) && (novo_evento == teste))
          {
             link_sendo_testado[0] = token;
             link_sendo_testado[1] = aux->v;

             if (link_falho[1] == link_sendo_testado[1])
                dissemina_mensagem(link_sendo_testado[0],
                                   link_falho[1]);
             novo_evento = testa_nodo(link_sendo_testado[0],
                                      link_sendo_testado[1],relogio);

             /* Falha de um nodo - NODO2F */
           if ((relogio >= relogiof) &&
               ((link_sendo_testado[0]==nodo1f) ||
                (link_sendo_testado[1]==nodo1f)) &&
               ((link_sendo_testado[0]==nodo2f) ||
                (link_sendo_testado[1]==nodo2f)))
          {
```

```
            schedule(falha,relogiof,nodo2f);
            novo_evento = dissemina;
         }
         aux = aux->next;
      }
      if (novo_evento == teste)
      {
         if (token >= V)
         {
               token = 1;
               relogio = relogio + 30.0;
         }
         else
             token ++;

            /* escalona o proximo teste */
         printf("\n"); fprintf(out,"\n");
         schedule(teste,relogio,token);
      }
      else
      {
         de_quem_recebeu[token] = link_falho[1];
         receberam[token] = 1;
         printf("\n\n\tE V E N T O   D I S S E M I N A");
         fprintf(out,"\n\n\tE V E N T O   D I S S E M I N A");
         printf("\n\n\tT=%5.1f; LINK %d-%d FALHO \n",relogiof,
                      nodo1f,nodo2f);
         fprintf(out,"\n\n\tT=%5.1f; LINK %d-%d FALHO \n",relogiof,
                     nodo1f,nodo2f);

         printf("\nT=%5.1f; NODO %d DESCOBRE NOVO ",relogio,token);
         printf("EVENTO LINK %d-%d FALHO\n",nodo1f,nodo2f);
         fprintf(out,"\n\tT=%5.1f; NODO %d DESCOBRE NOVO ",
                     relogio,token);
         fprintf(out,"EVENTO LINK %d-%d FALHO\n",nodo1f,nodo2f);
         schedule(dissemina,relogio,token);
         relogio++;
      }
      sair = TRUE;
      break;
   }

      case dissemina :
      {
         if (sair)
         {
            aux = nodo[token].adj_node[token].next;
            while (aux != z)
            {
               link_sendo_testado[0] = token;
               link_sendo_testado[1] = aux->v;
               if (((link_sendo_testado[0] == link_falho[0]) &&
                    (link_sendo_testado[1] == link_falho[1])) ||
                   ((link_sendo_testado[0] == link_falho[1]) &&
                    (link_sendo_testado[1] == link_falho[0])))
                  aux = aux->next;
               else
               {
                  if ((de_quem_recebeu[token] != aux->v) &&
                      (de_quem_recebeu[aux->v] != token))
                  {
```

```
            if (disseminaram[token]==0)
            {
                printf("\nT=%5.1f; nodo %d ",relogio,aux->v);
                printf("descobre evento <-- nodo %d",token);
                fprintf(out,"\nT=%5.1f; nodo %d ",
                            relogio,aux->v);
                fprintf(out,"descobre evento <-- nodo %d",
                            token);
                if (receberam[aux->v] == 0)
                {
                   dissemina_mensagem(token,link_falho[1]);
                   receberam[aux->v] = 1;
                   num_msg++;
                   if (de_quem_recebeu[aux->v] == 0)
                       de_quem_recebeu[aux->v] = token;
                }
                else
                {
                    redundante++;
                    printf(" **** REDUNDANTE");
                    fprintf(out," **** REDUNDANTE");
                 }
                 disseminam_mesmo_tempo[a] = aux->v;
                 a++;
              }
           }
           aux=aux->next;
       }

       if ((!dissemina2) && (token==nodo2f) &&
            (link_sendo_testado[1]==nodo1f))
       {
           dissemina2 = TRUE;
           token2 = nodo2f;
           relogiof = relogio;
           de_quem_recebeu2[token2] = nodo1f;
           printf("\n\n\tT=%5.1f; NODO %d DESCOBRE NOVO ",
                        relogio,token2);
           printf("EVENTO LINK %d-%d FALHO\n",nodo2f,nodo1f);
           fprintf(out,"\n\n\tT=%5.1f; NODO %d DESCOBRE NOVO ",
                            relogio,token2);
           fprintf(out,"EVENTO LINK %d-%d FALHO\n",nodo2f,
                        nodo1f);
       }
    }
    disseminaram[token] = 1;
    sair = todos_disseminaram_msg(disseminaram);
    if (sair)
    {
      if (disseminam_mesmo_tempo[b] != 0)
      {
         token = disseminam_mesmo_tempo[b];
         b++;
      }
      else
     {
        disseminam_mesmo_tempo[a] = 0;
        a++;
        relogio++;
        printf("\n"); fprintf(out,"\n");
        b++;
```

```
                token = disseminam_mesmo_tempo[b];
            }
            if (token != 0)
               schedule(dissemina,relogio,token);
            else
               sair = FALSE;
         }
         else
         {
            relogio2 = ++relogiof;
            schedule(dissemina,relogio2,token2);
            dissemina2=TRUE;
            acabou=TRUE;
            sair=FALSE;
         }
      }
/* Disseminacao Fase II - O segundo vertice do enlace falho tambem
   dissemina a mensagem contendo a falha v2-v1 */
      if (acabou)
      {
         aux2 = nodo[token2].adj_node[token2].next;
         while (aux2 != z)
         {
            link_sendo_testado2[0] = token2;
            link_sendo_testado2[1] = aux2->v;
            if (((link_sendo_testado2[0] == link_falho2[0]) &&
                 (link_sendo_testado2[1] == link_falho2[1])) ||
                ((link_sendo_testado2[0] == link_falho2[1]) &&
                 (link_sendo_testado2[1] == link_falho2[0])))
               aux2 = aux2->next;
            else
            {
               if ((de_quem_recebeu2[token2]!=aux2->v) &&
                   (de_quem_recebeu2[aux2->v]!=token2))
               {
                  if (disseminaram2[token2]==0)
                  {
                     printf("\nT=%5.1f; nodo %d ",relogio2,aux2->v);
                     printf("descobre evento <-- nodo %d (2a. msg)",
                             token2);
                     fprintf(out,"\nT=%5.1f; nodo %d ",relogio2,
                             aux2->v);
                     fprintf(out,"descobre evento <-- nodo
                             %d (2a. msg)",token2);
                     if (receberam2[aux2->v]==0)
                     {
                        dissemina_mensagem(token2,link_falho2[1]);
                        receberam2[aux2->v] = 1;
                        num_msg++;
                        if (de_quem_recebeu2[aux2->v] == 0)
                           de_quem_recebeu2[aux2->v] = token2;
                     }
                     else
                     {
                            redundante++;
                            printf("  **** REDUNDANTE");
                            fprintf(out,"  **** REDUNDANTE");
                     }
                     disseminam_mesmo_tempo2[aa] = aux2->v; aa++;
                  }
```

```
                    }
                    aux2=aux2->next;
                }
            }
            disseminaram2[token2] = 1;
            sair2 = todos_disseminaram_msg(disseminaram2);
            if (sair2)
            {
               if (disseminam_mesmo_tempo2[bb] != 0)
               {
                  token2 = disseminam_mesmo_tempo2[bb];
                  bb++;
               }
               else
               {
                  disseminam_mesmo_tempo2[aa] = 0;
                  aa++;
                  relogio2++;
                  printf("\n"); fprintf(out,"\n");
                  bb++;
                  token2 = disseminam_mesmo_tempo2[bb];
               }
               if (token2 != 0)
                  schedule(dissemina,relogio2,token2);
               else
                  sair2 = FALSE;
            }
         }
         break;
      }

          case falha:
          {
             link_falho[0] = nodo1f;
             link_falho[1] = nodo2f;
             link_falho2[0] = nodo2f;
             link_falho2[1] = nodo1f;
             break;
          }
      }
   }
   printf("\n\n\nRELATORIO FINAL:");
   fprintf(out,"\n\n\nRELATORIO FINAL:");

   printf("\n\t\tTotal de mensagens: %d",num_msg+redundante);
   fprintf(out,"\n\t\tTotal de mensagens: %d",num_msg+redundante);

   printf("\n\t\tNumero de mensagens redundantes: %d",redundante);
   fprintf(out,"\n\t\tNumero de mensagens redundantes: %d",redundante);

   printf("\n\t\tTempo para realizar o diagostico: %5.1f ",
                  relogio2-30.0);
   printf("unidades de tempo\n\n");
   fprintf(out,"\n\t\tTempo para realizar o diagnostico: %5.1f ",
                     relogio2-30.0);
   fprintf(out,"unidades de tempo\n\n");

} /* Fim do programa */
```