Um Algoritmo de Diagnóstico Distribuído para Redes de Topologia Dinâmica

Leandro Pacheco de Sousa

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

leandrops@inf.ufpr.br

Abstract. *Computer networks have been used in many different fields. This networks are getting larger and more complex, creating the need for automated tools for network management. Failure diagnosis is an important aspect of network management systems. This work aims to design an algorithm for distributed failure diagnosis on networks with dinamic topologies. The algorithm will be based upon the algorithm presented on [Sousa 2006]. The modification of the algorithm can be divided in two steps. First, the algorithm must be modified to work correctly in the case of network partitioning. Second, the algorithm must be extended to work with the insertion and removal of nodes during its execution. The latter is a considerably more complex task, since one of the premises of the base algorithm is that the network topology is static and every node knows it when the execution starts. When the algorithm is defined, it will be implemented and tested using the ns-2 simulator.*

Resumo. *As redes de computadores tem sido utilizadas nas mais diversas áreas. Estas redes têm se tornado cada vez maiores e mais complexas. Isto gera a necessidade da automação na gerência de redes. Um aspecto importante da gerência de redes é o diagnóstico de falhas. Este trabalho tem como objetivo desenvolver um algoritmo para diagnóstico de falhas em redes de topologia dinâmica. Este algoritmo será baseado no algoritmo apresentado em [Sousa 2006]. A modificação do algoritmo original pode ser dividida em dois passos. Primeiro, o algoritmo precisa ser modificado para funcionar corretamente no caso de particionamento na rede. Segundo, o algoritmo precisa ser extendido para suportar a inserção e remoção de nós durante sua execução. Esta última é uma tarefa considerávelmente mais complexa, pois uma das premissas do algoritmo original é que a topologia é estática e todo nó tem conhecimento da mesma quando a execução do algoritmo inicia. Após a definição do algoritmo, ele será implementado e testado no simulador ns-2.*

1. Introdução

Os Sistemas de gerência de redes permitem controlar e monitorar os elementos de uma rede. Uma função destes sistemas é a gerência de falhas. Os algoritmos de diagnóstico distribuído podem ser utilizados como base dos sistemas de gerência de falhas. O algoritmo *Distributed Network Reachability* [Duarte and Weber 2003, Weber et al. 2006], ou simplesmente DNR, é um algoritmo de diagnóstico em distribuído para redes de topologia arbitrária. Neste algoritmo, um enlace pode assumir 3 estados, falho, não-falho e

inatingível, que é quando este não é alcançável no estado atual da rede. Nós podem assumir apenas dois estados, sem-falha ou inatingível. Isto ocorre devido à ambiguidade das falhas em redes de topologia arbitrária. É impossível determinar se um nó está falho ou não se todos os seus enlaces não estiverem respondendo. O algoritmo consiste de três fases. Na fase de testes, são descobertas falhas e recuperações de nós ou enlaces, o que chamamos de eventos. Com a descoberta de um novo evento, é iniciada a fase de disseminação. Na fase de disseminação a informação sobre o novo evento é distribuída para toda a rede. A ultima fase é chamada fase de diagnóstico. A fase de diagnóstico pode ser iniciada por qualquer nó sem-falha, a qualquer momento. Nesta fase é feito o cálculo de alcançabilidade da rede através do uso de um algoritmo de conectividade em grafos. Uma das premissas do algoritmo DNR é o conhecimento prévio da topologia da rede. A topologia é estática e todos os nós já conhecem a mesma quando o algoritmo se inicia.

Em [Sousa 2006] é apresentada a implementação de uma variação do algoritmo DNR. Esta implementação é feita com o uso do simulador *ns-2*[ns2]. As diferenças entre o algoritmo implementado e o DNR estão na fase de disseminação. A disseminação no algoritmo implementado é feita com o uso de uma técnica de inundação. Outra diferença no algoritmo está no fato do mesmo não tratar do caso de particionamento e posterior recuperação da rede. Este trabalho propõe a modificação deste algoritmo adicionando o suporte ao particionamento na rede e a inclusão e remoção de nós durante a execução. O artigo está organizado da seguinte maneira. Na seção 2 são descritos os problemas que tentamos solucionar. Na seção 3 apresentamos nossas soluções propostas. Na seção 4 é descrita a nossa implementação do algoritmo no simulador. A seção 5 apresenta resultados de testes realizados. Por fim, na seção 6 apresentamos nossa conclusão.

2. Problema

O algoritmo de diagnóstico proposto em [Sousa 2006] possui duas limitações. A primeira é que este não leva em conta o caso de particionamento na rede. A segunda limitação é que o algoritmo precisa de conhecimento prévio sobre a topologia da rede. Quando o algoritmo se inicia, todos os nós possuem uma tabela contendo a topologia da rede e esta topologia se mantém estática durante a execução do algoritmo. Este trabalho tem como objetivo modificar o algoritmo para remover estas duas limitações.

2.1. Particionamentos na Rede

Em redes de topologia arbitrária, falhas de nós ou enlaces podem causar o particionamento na rede. Isto pode gerar problemas na execução do algoritmo em questão. Durante o particionamento cada componente conexo da rede continua executando o algoritmo normalmente. Quando um evento de recuperação que reconecte estes componentes ocorrer, podem haver nós com informações desatualizadas. Para resolver este problema, as informações contidas nos nós de cada componente conexo precisam ser repassadas para o resto da rede. Esta modificação estará restrita à fase de disseminação.

2.2. Topologia Dinâmica

Tanto no algoritmo proposto em [Sousa 2006] quanto no algoritmo DNR, existe a premissa de que todos os nós possuem conhecimento sobre a topologia da rede quando o algoritmo inicia. Esta topologia se mantém estática durante toda a execução do algoritmo. A segunda modificação proposta ao algoritmo em questão é adicionar o suporte

à topologias dinâmicas. Durante a execução do algoritmo, poderia ocorrer a inserção de novos nós ou novos enlaces e também a remoção de nós ou enlaces existentes. Esta é uma modificação relativamente complexa, pois haverão modificações nas três fases do algoritmo, testes, disseminação e diagnóstico.

Para a inserção de novos nós, é preciso adicionar ao algoritmo uma maneira de um novo nó enviar a informação de sua entrada no grupo para todos os seus vizinhos. Esta informação precisa então ser distribuída para o resto da rede. Para o caso de um novo enlace, os nós das extremidades do enlace devem iniciar a disseminação na rede. Isto já pode gerar alguns problemas que devem ser tratados. Uma inserção pode ocorrer durante um particionamento na rede. Nós de outro componente conexo não ficarão sabendo da inserção do novo nó ou enlace. A solução para este problema está relacionada à primeira modificação proposta. Quando ocorrer a detecção de uma recuperação de enlace, os nós das extremidades enviam informação sobre a estrutura atual da rede.

Para a remoção de nós, uma maneira de tratar o problema seria com o nó que irá sair do grupo enviando uma mensagem para seus vizinhos informando sobre sua saída. Talvez seja necessário o uso de *acks* para esta informação, já que pode existir o caso onde todos os enlaces deste nó falhem simultanêamente durante o envio da informação. Na remoção de enlaces, uma das pontas precisa iniciar o processo de remoção, avisando para todos os vizinhos sobre a remoção, inclusive o vizinho ligado ao enlace em questão. Em ambos os casos, podem surgir vários problemas. Um deles é o caso da remoção do nó ou enlace provocar o particionamento da rede. Neste caso, se um dos componentes conexos da rede não receber a informação durante a remoção, ele jamais irá receber esta informação. Na figura 1 estes casos são ilustrados. Talvez seja interessante que a remoção de um nó seja implementada com a remoção de enlaces. Assim, se um nó quiser sair da rede, ele primeiro inicia a remoção de todos os seus enlaces. Quando isto ocorrer, ele pode se desligar da rede. Da mesma maneira que com inserções, é preciso considerar uma remoção em todas as fases do algoritmo, pois podem haver consequências não previstas.

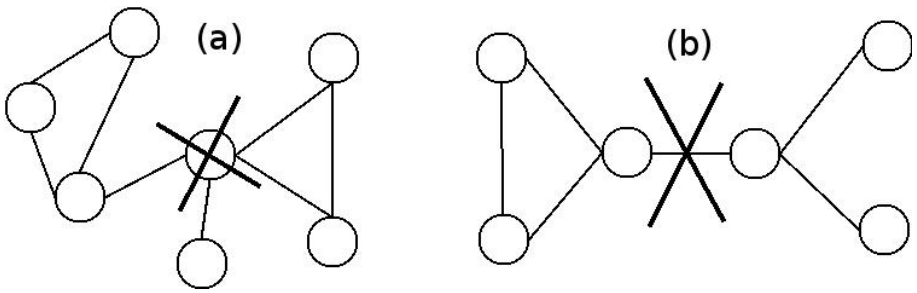

Figura 1. Remoção de nó(a) e enlace(b) causando particionamento

3. Soluções Propostas

3.1. Problema do Particionamento na Rede

A solução proposta para o caso do particionamento está restrita à fase de disseminação. Toda vez que houver uma recuperação de enlace que conecte dois componentes da rede, os dois nós do enlace enviam mensagens de disseminação sobre toda informação da rede que eles possuem. Este comportamento visa atualizar os dois componentes sobre novos eventos ocorridos durante o particionamento. Neste trabalho, foi decidido por fazer esta

disseminação em toda recuperação de enlace, evitando o uso de algoritmos de detecção de particionamento em grafos e tendo em vista que as mensagens de disseminação são relativamente pequenas.

3.2. Problema da Inserção e Remoção de Nós e Enlaces

O algoritmo proposto em [Sousa 2006], assim como o DNR, se utiliza de uma estrutura de evento, que identifica um enlace e possui um *timestamp*. Este timestamp é utilizado para ordenação de eventos e também para identificação do estado do enlace. Um timestamp de par identifica um enlace não-falho, e um número ímpar identifica um enlace falho. Em nossa solução decidimos por ampliar a estrutura com a inserção de um campo adicional de estado do enlace. Um enlace pode estar falho, não-falho ou ter sido removido. O motivo desta modificação está na solução para o problema do particionamento. Se ela não for repassada quando houver uma recuperação na rede, informações sobre enlaces removidos não serão repassadas para o outro componente conexo da rede. Assim, enlaces que existiam e foram removidos continuam sendo armazenados pelos nós. O timestamp é utilizado somente para a ordenação das mensagens.

Para a inserção e remoção de um enlace, foram definidos três novos tipos de mensagem, conexão, desconexão e confirmação de conexão/desconexão. Elas serão chamadas respectivamente de CONNECT, DISCONNECT e CONNACK. Para criar um enlace entre dois nós, uma das pontas envia uma mensagem de CONNECT para a outra ponta. Esta, quando recebe a mensagem, envia um CONNACK de volta para o vizinho e faz a disseminação do novo enlace para todos seus vizinhos. Quando o CONNACK é recebido pelo nó iniciador, ele começa os testes no enlace e dissemina o evento. Vale lembrar que este enlace se refere à um enlace ao nível do algoritmo de diagnósto, uma conexão em nível mais baixo já existia. Da mesma maneira que no caso da recuperação de enlaces, a criação de um novo enlace pode causar a junção de duas componentes conexas. Logo, os dois nós envolvidos devem enviar um ao outro toda informação que possuem. A remoção de enlaces funciona de maneira semelhante à inserção. Um nó inicia o processo enviando um DISCONNECT para seu vizinho, atualizando o estado do enlace como removido e fazendo a disseminação do evento. Quando um nó recebe o DISCONNECT, ele responde com um CONNACK, atualiza o estado do enlace como removido e faz a disseminação do evento. Os testes são interrompidos imediatamente. A inserção e remoção de nós não é implementada diretamente. Elas acontecem de maneira indireta, com a criação de enlaces entre nós, ou com a remoção de todos os enlaces do nó, respectivamente.

4. Implementação

Em [Sousa 2006] além da definição do algoritmo, também é descrita uma implementação do mesmo no simulador *ns-2*. O *ns-2* é um simulador de eventos discretos bastante utilizado na área de pesquisa em redes de computadores. A implementação descrita propõe a criação de um novo agente e uma nova aplicação para o *ns-2*. Estes são chamados respectivamente de *DNRAgent* e *DNRApp*. Cada *DNRAgent* está ligado a um nó e possui uma conexão com outro *DNRAgent* ligado a outro nó. Esta conexão forma o canal de comunicação entre dois nós, caracterizando um enlace. Desta maneira, o número de agentes ligados à um nó qualquer é igual ao número de nós vizinhos. Os agentes são utilizados apenas para envio e recebimento de mensagens do algoritmo. As aplicações *DNRApp* são os componentes que executam o algoritmo. Cada nó possui exatamente uma *DNRApp*,

que está ligada à todos os agentes do nó. Cada aplicação executa o algoritmo de maneira independente das outras e possui informação sobre a topologia e o estado de toda a rede. A figura 2 mostra a configuração do simulador para uma topologia exemplo.

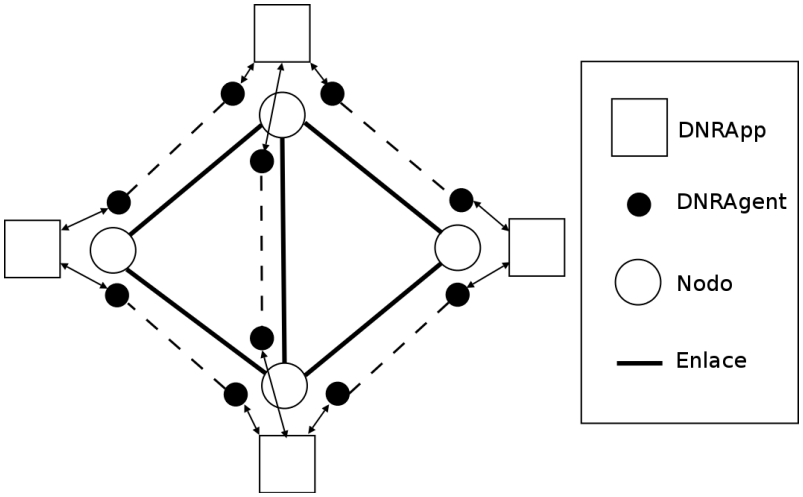

Figura 2. Configuração de nós, agentes e aplicações para uma topologia exemplo

Este trabalho acrescenta as modificações propostas anteriormente à implementação existente.

4.1. Suporte ao Particionamento

Para o suporte ao particionamento na rede, é feita uma modificação na disseminação de eventos de recuperação. Quando um nó detecta uma recuperação, ele envia toda informação que possui sobre a rede para a outra ponta do enlace recuperado. Isto foi implementado com o envio de uma mensagem de disseminação relacionada a cada enlace que o nó conhece. Esta modificação é restrita à aplicação *DNRApp*.

4.2. Suporte à Remoção e Inserção de Enlaces

Para a implementação desta solução, foram necessárias modificações no pacote DNR, no agente *DNRAgent* e na aplicação *DNRApp*.

A primeira modificação ocorre na estrutura do pacote DNR. Este deve conter um campo a mais, chamado *status*. Este campo está relacionado à informação de estado que foi adicionada à estrutura de um evento.

No *DNRAgent*, foram adicionados três métodos, *sendConnect()* , *sendDisconnect()* e *sendConnAck()*. O funcionamento deles é trivial, eles simplesmente enviam um pacote do tipo especificado pelo nome da função.

As modificações mais complexas ocorreram na aplicação *DNRApp*. Quando uma aplicação inicia sua execução, ela não possui informação sobre nenhum enlace. Isto significa que cada nó do algoritmo é iniciado sem nenhum vizinho. Esta definição de vizinho está se referindo ao nível do algoritmo, mas devido ao funcionamento do *ns-2*, a estrutura da rede já está definida. Para criar um novo enlace, uma *DNRApp* precisa iniciar o processo enviando um pacote CONNECT para uma aplicação vizinha, com o uso de

um *DNRAgent*. Quando o enlace entre os dois nós é estabelecido, os nós trocam suas informações sobre a rede e os testes no enlace são iniciados. A partir disso, o funcionamento do algoritmo não se altera. Na remoção de um enlace, uma *DNRApp* envia um DISCONNECT para uma aplicação vizinha, iniciando o processo de remoção. Um enlace removido continua sendo armazenado pelas aplicações do algoritmo, apenas seu estado é alterado. Tanto a remoção quanto a inserção de enlaces são novos eventos, logo devem ser disseminados normalmente. Nossa implementação não tolera falhas que ocorrem no enlace durante o processo de criação ou remoção do mesmo.

5. Testes e Resultados

Apresentamos aqui os resultados de testes realizados em três topologias diferentes. Um dos objetivos dos testes foi verificar a corretude do algoritmo, com base nos dados contidos nos nós ao fim da execução. Outro objetivo foi obter uma relação entre número de mensagens enviadas e número de mensagens redundates para cada caso de teste. Uma mensagem é redundante quando ela contém um evento igual ou mais velho do que o evento contido no nó receptor.

5.1. Topologia 1

A topologia 1 pode ser observada na Figura 3. O primeiro teste executado nesta topologia

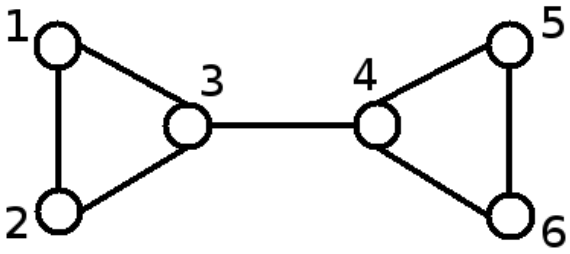

Figura 3. Configuração da Topologia 1

é o caso do particionamento e posterior recuperação da rede. A sequencia de eventos é a seguinte:

- Todos os enlaces são adicionados simultâneamente
- O enlace 3-4 falha
- Ocorre uma rodada de testes e disseminação
- Os enlaces 1-2 e 5-6 falham
- Ocorre uma rodada de testes e disseminação
- O enlace 3-4 se recupera
- Ocorre uma rodada de testes e disseminação

Ao final, como esperado, todos os nós possuíam a mesma informação, contendo os enlaces 1-2 e 5-6 como falhos, apesar de as falhas terem ocorrido enquanto a rede estava particionada. Neste caso, o número de mensagens foi 184 e o número de mensagens redundantes 56.

No segundo teste os eventos ocorridos são os mesmos. A diferença está no fato de o enlace 3-4 ser removido e novamente inserido ao invés de falhar e se recuperar. O resultado final foi o mesmo, e o número de mensagens foi 184 e o número de mensagens redundantes 58.

5.2. Topologia 2

Neste teste, a topologia utilizada foi a da Figura 4. Para o teste realizado nesta topologia,

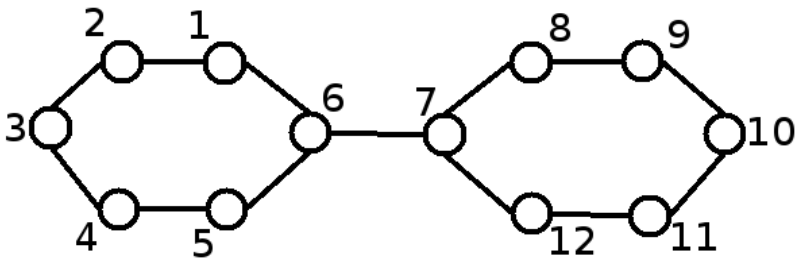

Figura 4. Configuração da Topologia 2

a sequencia de eventos é a seguinte:

- Todos os enlaces são adicionados simultâneamente
- O enlace 6-7 falha
- Ocorre uma rodada de testes e disseminação
- Os nós 3 e 10 falham
- Ocorre uma rodada de testes e disseminação
- O enlace 6-7 se recupera
- Ocorre uma rodada de testes e disseminação

Ao final da execução, todos nós possuiam a mesma informação como esperado. O numero de mensagens enviadas foi 441, e o número de mensagens redundantes foi 104.

5.3. Topologia 3

A topologia 3 pode ser observada na Figura 5.

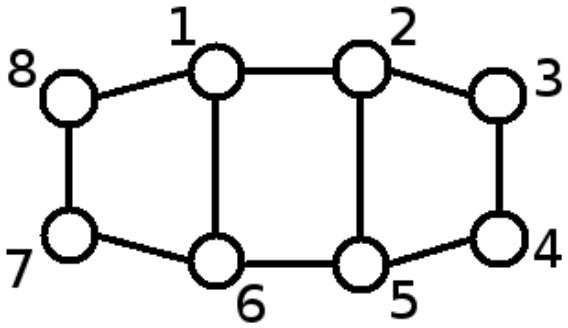

Figura 5. Configuração da Topologia 3

Neste teste, os enlaces não são adicionados simultâneamente como anteriormente, existe um tempo entre a inserção de um enlace e a inserção do próximo. Isso faz com que os intervalos de testes nos enlaces aconteçam em tempos bastante diferentes. Após todos enlaces serem inseridos e a disseminação ser terminada, os enlaces 1-2 e 6-5 são removidos. Depois, após algumas rodadas de testes por alguns enlaces, os enlaces 8-7

e 3-4 falham. São executadas as rodadas de teste e disseminação, e a execução termina. Neste caso, devido ao particionamento na rede, os nós de uma componente não sabem sobre a falha na outra componente. O número de mensagens enviadas foi 257, e o numero de mensagens redundantes 88.

5.4. Análise dos Resultados

Com base nos testes aqui descritos, juntamente com alguns outros não citados aqui, pudemos observar que o funcionamento do algoritmo está correro. Como dito anteriormente, nosso algoritmo não garante o funcionamento correto em caso de falha de um enlace enquanto ele está sendo inserido ou removido.

De acordo com os testes aqui demonstrados, a porcentagem de mensagens redundantes foi de aproximadamente 30%. Este valor seria provavelmente bastante inferior em uma situação real. Isto porque, nos testes realizados, são detectados novos eventos praticamente em toda rodada de testes. Devido ao pequeno tamanho das mensagens de diagnóstico, o elevado número de mensagens enviadas não é tão relevante.

6. Conclusão

Apresentamos neste trabalho duas propostas de modificação ao algoritmo de diagnóstico distribuído descrito em [Sousa 2006]. Estas modificações buscavam eliminar restrições do algoritmo original. A primeira modificação teve por objetivo tornar o algoritmo tolerante a particionamentos na rede. A segunda modificação foi proposta para que o algoritmo pudesse executar em redes de topologia dinâmica, com inserção e remoção de nós e enlaces. Também fizemos a implementação destas modificações no simulador *ns-2* e expusemos uma breve descrição sobre a mesma. Foram realizados testes, onde foi verificada a corretude do algoritmo e também a porcentagem de mensagens redundantes, que foi de aproximadamente 30% do total de mensagens enviadas nos casos de teste.

Referências

The network simulator - ns2. *http://www.isi.edu/nsnam/ns/*, acessado em Maio de 2007.

Duarte, E. P. and Weber, A. (2003). A distributed network connectivity algorithm. In *Proc. IEEE/ISADS'03*, Pisa.

Sousa, L. P. (2006). Simulação de um algoritmo de diagnóstico baseado em token-test com disseminação por inundação. Monografia apresentada ao Curso de Bacharelado em Ciência da Computação.

Weber, A., Duarte, E. P., and Fonseca, K. V. O. (2006). An optimal test assignment for monitoring general topology networks. In *The 7th IEEE Latin Amerincan Test Workshop (IEEE LATW'2006)*, Buenos Aires, Argentina.