Computação Distribuída de Conectividade de Redes de Topologia Arbitrária

Andréa Weber[1] **, Elias Procópio Duarte Jr.**[1]

[1]Departamento de Informática – Universidade Federal do Paraná
Cx.Postal 19081 Curitiba 81531-990 PR Brasil
e-mail: {andrea,elias}@inf.ufpr.br

Abstract. The *Distributed Network Connectivity* algorithm allows every node in a general topology network to determine which portions of the network are reachable and unreachable. The algorithm consists of three phases: test, dissemination, and connectivity computation. During the testing phase each link is tested by one of the adjacent nodes at alternating testing intervals. Upon the detection of a new unresponsive link, the tester starts the dissemination phase, in which a distributed breadth-first tree is employed to inform the other connected nodes about the event. At any time, any working node may run the third phase, in which a graph connectivity algorithm shows the network connectivity. We prove bounds on the worst case latency of the algorithm. Simulation results of the dissemination of one event are presented for a number of network topologies, and compared to other algorithms.

Resumo. O algoritmo *Distributed Network Connectivity* permite que cada nodo em uma rede de topologia arbitrária determine que partes da rede são atingíveis e que partes são inatingíveis. O algoritmo consiste de três fases: teste, disseminação, e cálculo da conectividade. Durante a fase de teste, cada enlace é testado por um dos nodos adjacentes, em intervalos alternados. Ao detectar a falha de um enlace, o testador dá início à fase de disseminação, na qual uma árvore distribuída de busca em largura é utilizada para informar sobre o evento aos outros nodos conectados. A qualquer tempo, cada nodo sem-falha pode executar a terceira fase, na qual um algoritmo de conectividade de grafos mostra a conectividade da rede. Limites para o pior caso de latência do algoritmo são provados. Resultados de simulação da disseminação de um evento de falha são apresentados para várias topologias de rede e são comparados aos de outros algoritmos.

Palavras chave: Diagnose Distribuída, Conectividade de Grafos, Particionamento, Redes de Topologia Arbitrária, Algoritmos Distribuídos, Gerenciamento de Redes.

1. Introdução

Organizações e indivíduos dependem cada vez mais do correto funcionamento de sistemas computacionais e de rede. Dada uma topologia, é importante determinar a qualquer instante de tempo quais porções da rede estão atingíveis e quais porções estão inatingíveis. Neste trabalho, um novo algoritmo para o cálculo distribuído de conectividade de redes de topologia arbitrária, chamado ”Distributed Network Connectivity” (DNC), apresenta uma abordagem tolerante a falhas para monitoração *on-line* de redes, e pode ser utilizado por qualquer aplicação baseada na conectividade do sistema, tais como sistemas de gerenciamento de redes [1] os quais apresentam ao usuário um mapa que mostra o estado da rede.

Numa rede de topologia arbitrária não há necessariamente um enlace de comunicação entre cada par de nodos. Ambos nodos e enlaces podem estar tanto falhos como sem-falha a um dado instante de tempo. Considera-se falhas do tipo *crash* e uma falha pode particionar a rede. Considerando um par de nodos conectados por um enlace, se os testes executados em um nodo pelo outro determinam que o enlace não está respondendo, é impossível determinar se é o nodo testado ou o enlace que está falho. Deste modo, falhas em uma rede de topologia arbitrária são ditas ambíguas [4].

Embora seja baseado em resultados prévios de diagnóstico em nível de sistema [3] de redes de topologia arbitrária, o algoritmo apresenta uma perspectiva diferente da diagonse propriamente dita. Em [6] Bagchi e Hakimi introduziram um algoritmo para diagnóstico em nível de sistema de redes de topologia arbitrária. O algoritmo não pode ser usado para monitoração contínua da conectividade de uma rede porque o mesmo é executado *off-line*. Em [7] Bianchini e Buskens introduziram e avaliaram por simulação o algoritmo *Adapt*, o qual pode ser executado *on-line*: os nodos constroem de forma distribuída um grafo de testes correspondente a uma árvore; quando um dado nodo se torna falho, uma nova fase é iniciada na qual outros nodos reconectam o grafo de testes. O algoritmo emprega um procedimento distribuído que utiliza uma estratégia sequencial de disseminação de eventos.

Rangarajan, Dahbura e Ziegler [8] introduziram o algoritmo RDZ de diagnóstico em nível de sistema para redes de topologia arbitrária o qual pode ser executado *on-line*. O algoritmo constrói um grafo de testes que garante um número ótimo de testes, isto é, cada nodo tem apenas um testador. Além disso, apresenta a melhor latência de diagnóstico possível através da utilização de uma estratégia paralela de disseminação. Sempre que um nodo detecta um evento, ele envia a informação de diagnóstico para todos os seus vizinhos, os quais, a seu turno enviam a informação a todos os seus vizinhos, e assim sucessivamente. Embora o algoritmo RDZ apresente a melhor latência de diagnóstico possível, e o melhor número possível de testadores por nodo, ele somente garante a eventual diagnose de uma configuração de falha específica, chamada pelos autores de jellyfish [8].

O algoritmo *Distributed Network Connectivity* (DNC) introduzido neste trabalho é baseado no algoritmo *Non-Broadcast Network Diagnosis* (NBND) [4, 5, 9]. O algoritmo é estruturado em três fases: teste, disseminação e diagnóstico. Durante a fase de testes, cada enlace é testado por um de seus nodos adjacentes em intervalos de teste alternados. Ao detectar um novo enlace não respondendo, o testador inicia a fase de disseminação, na qual uma árvore distribuída de busca em largura é empregada para informar aos outros nodos conectados sobre o evento. A qualquer instante de tempo, qualquer nodo sem-falha pode rodar a terceira fase, na qual um algoritmo de conectividade em grafos mostra a conectividade completa da rede. O algoritmo permite a ocorrência de eventos dinâmicos, isto é. durante a fase de disseminação, novos eventos podem ocorrer e sua disseminação permanece garantida.

O resto do artigo é organizado como segue. Na seção 2 o algoritmo é descrito. A seção 3 contém a especificação do algoritmo. Na seção 4 os limites do pior caso de latência do algoritmo são provados. A seção 5 apresenta resultados de simulação da disseminação de um evento em várias topologias de rede; os resultados são comparados aos de outros algoritmos. A seção 6 conclui o trabalho.

2. Descrição do Algoritmo

Nesta seção é introduzido o novo algoritmo para cálculo de conectividade de redes de topologia arbitrária. O algoritmo consiste de três fases: testes, disseminação da informação sobre novos eventos e cálculo local da conectividade. Nodos executam testes para determinar o estado de enlaces adjacentes. Se não há resposta a um dado teste sobre um enlace, o nodo testador não é capaz de determinar se o enlace testado ou o nodo conectado por aquele enlace não está operando apropriadamente. Se não há nenhuma resposta a testes executados sobre todos os enlaces de um nodo, então o nodo é considerado *inatingível*. Nodos, portanto, podem estar nos estados *sem-falha* ou *inatingível*, e enlaces podem estar em um de três estados: *sem-falha*, *silencioso* ou *inatingível*. Um nodo considera um enlace como inatingível quando o enlace não é adjacente a nenhum outro nodo atingível sem-falha.

Root	Tester 1	Tested 1	Tstmp 1	Tester 2	Tested 2	Tstmp 2	. . .	Tester i	Tested i	Tstmp i

Figura 1: Formato da mensagem DNC

Cada enlace é testado a cada intervalo de testes. Há um testador por enlace por intervalo de testes. O algoritmo, portanto, requer o número mínimo de testes para qualquer topologia de rede. O algoritmo utiliza uma estratégia de testes baseada em *token* [9]: ambos os nodos conectados por um enlace executam testes sobre aquele enlace em intervalos alternados. Os testes utilizados são também ditos *two-way tests*, no sentido em que, quando um nodo executa um teste sobre um enlace, não somente o nodo testador determina o estado do nodo testado, mas também o nodo testado determina o estado do testador.

Cada nodo mantém um *timestamp* que é um contador de estados para cada enlace no sistema [8]. Cada *timestamp* é inicialmente zero, e é incrementado a cada novo evento de falha (ou de recuperação) detectado no respectivo enlace. Isto permite a um nodo identificar mensagens redundantes. Uma mensagem é considerada redundante quando não é a primeira sobre um dado conjunto de eventos. Após um novo evento ser detectado, o testador propaga a informação sobre o mesmo a seus vizinhos. Cada nodo mantém a topologia completa da rede. A estratégia de disseminação paralela empregada é baseada em uma árvore distribuída de busca em largura.

Cada mensagem de diagnóstico, mostrada na figura 1, leva a seguinte informação: a raiz da árvore, e um conjunto de informações de eventos em enlaces que contém, para cada evento: (1) a identificação do nodo testador, (2) a identificação do nodo testado, e (3) o *timestamp* para o enlace testado. Cada nodo rodando o algoritmo mantém uma tabela de enlaces indexada pelos identificadores do enlace, contendo o *timestamp* para o enlace. Um *timestamp* par indica um enlace sem falha; um *timestamp* ímpar indica que o enlace está silencioso [8].

Cada nodo mantém o mesmo grafo representando a topologia da rede, a qual é atualizada a cada evento nas mensagens recebidas, isto é, enlaces silenciosos são removidos do grafo. Como cada nodo também é informado sobre a raiz da árvore quando do recebimento de dada mensagem, todos os nodos montam a mesma árvore de busca em largura ao disseminar aquela mensagem.

Considerando a árvore de busca em largura, após a disseminação ter sido iniciada, as novas informações sobre eventos são consideradas *pendentes*, até que cada nodo sem-falha no sistema confirme o recebimento daquela informação. Mensagens de confirmação são propagadas dos nodos-folha para a raiz. Sempre que um nodo é uma folha, após receber a mensagem de seu pai ele envia uma confirmação, chamada *ack*.

Após receber *acks* de todos os seus filhos, um nodo envia um *ack* a seu pai na árvore. A disseminação é completada quando a raiz recebe confirmações de todos os seus filhos.

2.1. Disseminando Múltiplos Eventos

Quando um novo evento ocorre e a disseminação do evento precedente ainda não foi completada, nodos devem realizar a disseminação de múltiplos eventos.

Seja uma “mensagem pendente” a mensagem de uma disseminação pendente. Seja “mensagem recebida” a mensagem recebida por um nodo. Ambas as mensagens podem conter novas informações uma com relação à outra. Uma mensagem contém informação nova se ela contém informação sobre um evento em um enlace que não está presente na outra mensagem, ou se ela possui informação sobre um enlace que está na outra mensagem, porém com *timestamp* maior.

Quando a mensagem recebida tém informação nova com respeito à mensagem pendente,

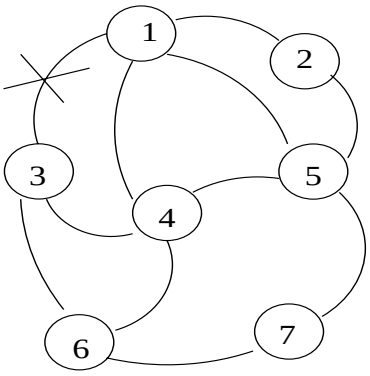

Figura 2: Uma topologia de exemplo. Enlace 1–3 se torna silencioso.

o nodo agrega a nova informação recebida à informação na disseminação pendente. Se a mensagem pendente tem nova informação com respeito à mensagem recebida, o nodo inicia uma nova árvore de disseminação. Se esse não é o caso, então o nodo simplesmente se insere na árvore de disseminação distribuída de acordo com a mensagem recebida. A disseminação precedente é abandonada quando a nova disseminação tem informação completa sobre todos os eventos. Se a mensagem recebida não possui informação nova com respeito à mensagem pendente, então a mensagem recebida é simplesmente descartada.

Quando a mensagem recebida e a mensagem pendente são exatamente as mesmas, elas vêm necessariamente de árvores diferentes, iniciadas por nodos diferentes. Nesse caso, o nodo tem que tomar parte em ambas as árvores de disseminação. Esta situação ocorre quando dois ou mais nodos são informados sobre o mesmo conjunto de eventos antes que a disseminação iniciada por um deles alcance os demais. Neste caso, nenhuma das árvores de disseminação pode ser abandonada, por não haver critérios para selecionar uma das mensagens e descartar as demais. Um exemplo é dado na seqüência.

2.2. Calculando a Conectividade

A qualquer tempo qualquer nodo sem-falha rodando o algoritmo pode computar localmente a conectividade da rede após remover da topologia os enlaces que estão no estado *silencioso*, isto é, que tém um *timestamp* ímpar.

2.3. Exemplos de Execução

Considere a topologia exemplo na figura 2. Esta subseção contêm uma descrição da execução do algoritmo, considerando um evento de falha em um enlace, e a subsequente disseminação da informação sobre o novo evento.

Considere a ocorrência de um evento de falha no enlace 1–3 no instante de tempo t_0. No próximo intervalo de testes após a ocorrência do evento, supõe-se que o nodo 1 é o nodo testador, e portanto detecta a falha. Então o nodo 1 inicia a fase de disseminação de forma a comunicar a informação sobre o evento a todos os demais nodos atingíveis. Ele o faz montando uma árvore de busca em largura com raiz em si mesmo, já levando em consideração o evento de falha. A árvore é ilustrada na figura 3 A.

A árvore de disseminação possui o nodo 2, nodo 4 e nodo 5 no segundo nível. No terceiro nível, o nodo 4 tem dois filhos: nodo 3 e nodo 6; o nodo 5 tem um filho: o nodo 7. Considere que decorre uma unidade de tempo para um nodo processar a mensagem de disseminação e para essa mensagem ser recebida pelos filhos do nodo. Portanto, uma unidade de tempo após o envio da mensagem de disseminação pelo nodo 1 ela chega aos nodos 2, 4 e 5 simultaneamente.

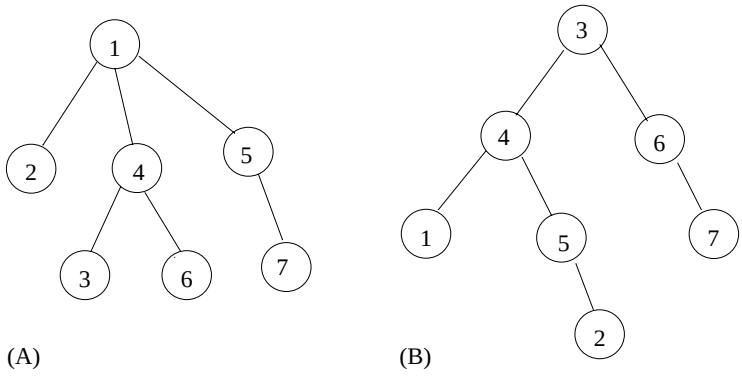

Figura 3: As árvores de disseminação iniciadas pelo nodo 1 (A) e nodo 3 (B).

Uma vez recebida a mensagem, se o nodo é uma folha ele deve enviar uma confirmação ao nodo pai, e se ele não é uma folha ele deve encaminhar a mensagem aos seus filhos na árvore, e enviar uma confirmação ao seu pai quando todos os seus filhos confirmaram a mensagem; se o nodo tem outro evento a informar, ele desiste da disseminação corrente e inicia uma nova que mantèm o conteúdo da disseminação pendente.

Neste exemplo de disseminação, o nodo 2 é uma folha na árvore de disseminação, de forma que a mensagem que ele recebe é confirmada na próxima unidade de tempo. Os nodos 4 e 5 disseminam a informação para seus filhos na próxima unidade de tempo, de tal forma que, quando a confirmação do nodo 2 chega ao nodo 1, mensagens de disseminação alcançam os nodos 3, 6 e 7 na base da árvore. Os nodos 6 e 7 são folhas, e portanto confirmam o recebimento da mensagem aos nodos 4 e 5, respectivamente. O nodo 3, entretanto, uma vez informado sobre a falha no enlace 1–3 testa seus enlaces adjacentes, e conclui que o enlace 3–1 também está falho. Portanto, esta informação deve ser disseminada. Para que isto seja feito, o nodo 3 monta outra árvore de disseminação, como ilustrado na figura 3 B.

Neste ponto há duas árvores de disseminação simultâneas. A primeira árvore de disseminação foi iniciada pelo nodo 1, e dissemina informação sobre o evento de falha no enlace 1–3, a qual é chamada primeira mensagem de disseminação. A primeira árvore de disseminação vai sendo gradualmente abandonada à medida que a segunda árvore de disseminação, que foi iniciada pelo nodo 3 e tem informação completa a respeito de ambos os eventos (enlace 1–3 está falho, enlace 3–1 está falho), toma seu lugar.

À medida que a execução continua, ao mesmo tempo em que as mensagens de confirmação dos nodos 6 e 7 chegam aos nodos 4 e 5 da primeira árvore de disseminação, novas mensagens são enviadas pelo nodo 3 na segunda árvore de disseminação. Os nodos 4 e 6 recebem estas mensagens. O nodo 4 envia a mensagem de disseminação aos nodos 1 e 5. Quando o nodo 6 recebe a segunda mensagem de disseminação, ele já completou sua tarefa na primeira árvore de disseminação. Então, ele envia a segunda mensagem de disseminação ao nodo 7, seu filho na segunda árvore. Isto ocorre simultaneamente ao envio da segunda mensagem de disseminação aos nodos 1 e 5 pelo nodo 4.

Ao receber informação sobre este segundo evento, o nodo 4 compara seu conteúdo com o conteúdo correspondente na mensagem pendente e percebe que esta mensagem contém a informação enviada pela primeira mensagem de disseminação além de dados novos. Então o nodo 4 decide pela disseminação da segunda mensagem, por ser vantajosa, e descarta a primeira mensagem. Isto ocorre com o envio de nova mensagem aos nodos 1 e 5.

No terceiro nível da segunda árvore de disseminação, os nodos 1 e 7 são folhas, as quais

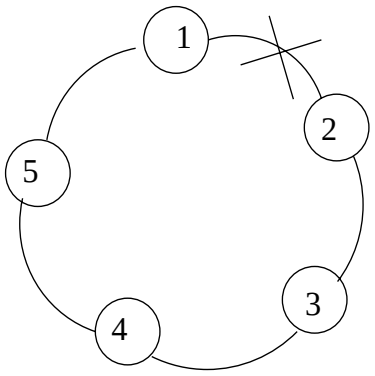

Figura 4: Uma topologia exemplo. Enlace 1–2 se torna silencioso.

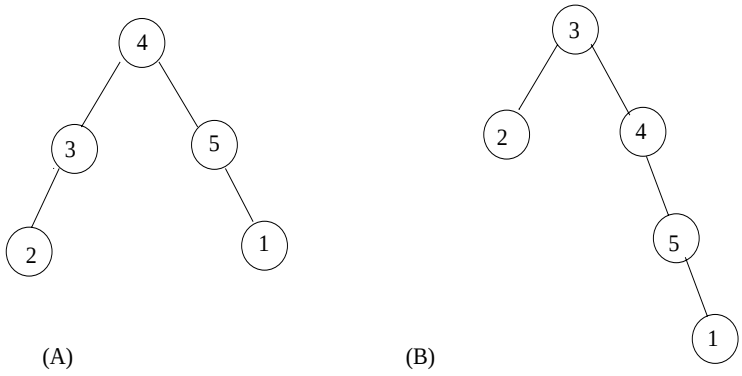

Figura 5: As árvores de disseminação iniciadas pelo nodo 1 (A) e nodo 2 (B).

confirmam as mensagens recebidas aos seus nodos-pai. Ao mesmo tempo em que estas confirmações sobem um nível na árvore, o nodo 5 envia a segunda mensagem ao nodo 2, seu filho na segunda árvore de disseminação, o qual envia uma confirmação após seu recebimento. Ao mesmo tempo em que o nodo 5 recebe a mensagem de confirmação do nodo 2, a raiz da árvore recebe a mensagem de confirmação do nodo 6. Na próxima unidade de tempo, o nodo 5 confirma a mensagem do nodo 2, e o nodo 4 faz o mesmo após receber a confirmação do nodo 5. Após isto, a raiz, no nodo 3, recebe a última confirmação e considera a disseminação concluída. Uma vez iniciada na primeira raiz no nodo 1, a disseminação como um todo se completa em 8 unidades de tempo.

Agora considere a topologia em anel na figura 4. Um evento de falha ocorre no enlace 1–2 no instante t_0. Após a detecção deste evento no intervalo de teste seguinte, o detector, nodo 1, inicia a disseminação desta informação. O nodo 5 recebe a mensagem de disseminação na próxima unidade de tempo. Considere que, a este tempo, o nodo 2 também detecta o evento de falha no enlace 2–1, e também inicia sua disseminação. No intervalo de tempo seguinte, ambos os nodo 3 e 4 recebem as mensagens vinda dos nodos 2 e 5, respectivamente.

No intervalo de tempo seguinte, ambos os nodos 3 e 4 recebem as mensagens vindas um do outro. Do ponto de vista desses nodos, ambas as mensagens recebidas contém nova informação considerando suas próprias mensagens pendentes. Portanto, o nodo 3 inicia a disseminação da mensagem (2–1, 1–2), e o nodo 4 inicia a disseminação da mensagem (1–2, 2–1) empregando as árvores mostradas na figura 5. Não é difícil perceber que ambas as mensagens contém informação sobre exatamente os mesmos eventos.

Neste caso, ambas as mensagens tém que ser disseminadas, pelo fato de que ambos os

Algoritmo *Distributed Network Connectivity* Executado pelo Nodo i

```
type Event = record
                Tester : NodeId;
                Tested : NodeId;
                Timestamp : Counter;
             end;
type Message = record
                 Root : NodeId;
                 Ack : Bit;
                 list of Event;
               end;
var Status : array of array of NodeId;
    RcvdMsg, AckMsg : Message;
    PendingMsg : list of Message;
    AckToReceive = {}, AckToSend = {} : list of set of NodeId;
    DisseminationId = 0 : LongInteger;
    Token, TokenTurn : list of boolean;
for all j,l such that link (j,l) exists
    Status[j,l] = 0
    Status[l,j] = 0
end for
for all j such that link i,j exists
    TokenTurn[j] = FALSE;
    if i > j
    then Token[j] = TRUE
    else Token[j] = FALSE
    end if
end for
```

Figura 6: Algoritmo DNC: declaração de variáveis.

nodos iniciaram suas novas árvores de disseminação ao mesmo tempo com o mesmo conteúdo, e é impossivel concordar em uma das árvores para ser descartada.

3. Especificação do Algoritmo

Nesta seção o algoritmo é especificado. A figura 6 contém a declaração de variáveis globais e sua inicialização. A figura 7 contém o algoritmo de testes baseado em *token*. A figura 8 contém o algoritmo de disseminação, o qual é baseado numa árvore de busca em largura. O código é comentado abaixo.

Inicialmente a estrutura de dados usada para manter informações sobre um evento (`Event`) é declarada. O algoritmo considera eventos em enlaces, nos quais um nodo detecta que um enlace sem-falha se tornou silencioso. Um evento, portanto, é descrito pelo identificador do nodo testador (`Tester`), pelo identificador do nodo testado (`Tested`) e pelo `Timestamp` correspondente. Uma mensagem (`Message`) consiste de uma lista de eventos precedida pela raiz (`Root`) da árvore de disseminação. Um bit de `Ack` é empregado para permitir que confirmações sejam comunicadas com a mesma estrutura de mensagem. Nenhuma identificação de mensagem é necessária, porque a lista de eventos junto com o identificador da raiz da árvore são suficientes para

Algoritmo *Distributed Network Connectivity* Executado pelo Nodo

```
Start&RunForever()
while TRUE do
   for each neighbor j
       CheckTestToken(j)
   end for
   waituntil Testing Interval expires
end while
|| Module: CheckTestToken(j)
  if Token[j]
  then Token[j] = FALSE;
       RunTest(j)
  else if TokenTurn[j]
       then Token[j] = TRUE;
            CheckTestToken(j);
            TokenTurn[j] = FALSE
       else TokenTurn[j] = TRUE
       end if
  end if
|| Module: RespondTest(j)
  receive test request from j; send test reply to j;
  if TokenTurn[j]
  then TokenTurn[j] = FALSE
  end if
  Token[j] = TRUE;
|| Module: RunTest(j)
  if (Status[i,j] MOD 2 != Test-Procedure(i,j))
  then Status[i,j]++;
       for all m AckToReceive_m = {}, AckToSend_m = {} end for all
       BFT-Disseminate (i, merge all pending messages + new event);
  end if
```

Figura 7: Algoritmo DNC: fase de testes.

tornar cada mensagem única.

Cada nodo mantém informação sobre o estado de cada enlace na rede (`Status`). O estado (`Timestamp`) é inicialmente zero. Um nodo pode receber dois tipos de mensagens: uma mensgem de disseminação (`RcvdMsg`) ou uma mensagem de confirmação (`AckMsg`). Após uma dada disseminação ser iniciada, e antes que a mesma termine ela é considerada pendente. Uma vez que um nodo pode tomar parte em mais de uma disseminação concorrente, cada nodo mantém uma lista de mensagens pendentes (`PendingMsg`). Para cada disseminação pendente um nodo mantém listas de indentificadores de nodos para os quais confirmações devem ser enviadas (`AckToSend`) ou dos quais confirmações devem ser recebidas (`AckToReceive`). Uma identificação local da disseminação (`DisseminationId`) é mantida para cada disseminação que um nodo inicia ou toma parte.

A cada intervalo de testes cada enlace é testado por um dos dois nodos os quais ele conecta. A figura 7 contém o procedimento de teste. Dois nodos adjacentes trocam um `Token` a cada intervalo de testes. `TokenTurn` é empregado por um dos nodos para forçar um teste quando

o outro nodo está falho. Quando um novo evento é detectado, a fase de disseminação é iniciada. Se já existe uma mensagem pendente, a informação sobre o novo evento é inserida na mensagem pendente para iniciar uma nova disseminação. A figura 8 contém o algoritmo distribuído de disseminação.

O módulo `BFT-Disseminate` recebe como parâmetros a raiz (`Root`) da árvore de busca em largura na qual a mensagem (`Message`) deve ser disseminada. Uma nova identificação para a mensagem (`DisseminationId`) é gerada. A árvore é montada com base na topologia atualizada, isto é, removendo do grafo os enlaces falhos. A mensagem é enviada a cada filho do nodo i na árvore. `AckToSend` e `AckToReceive` são atualizados apropriadamente.

Quatro diferentes situações podem ocorrer após um nodo receber uma mensagem de disseminação. (1) No caso das mensagens pendentes terem informações novas comparadas à mensagem recebida, e a mensagem recebida também ter informações novas comparadas às mensagens pendentes, uma nova disseminação é iniciada, usando o nodo corrente como raiz. A mensagem disseminada é a combinação das mensagens pendentes e da mensagem recebida. (2) No caso das mensagens pendentes terem informações novas comparadas à mensagem recebida, mas a mensagem recebida não ter nenhuma nova informação, a mensagem recebida é descartada. (3) No caso das mensagens pendentes não terem informações novas comparadas à mensagem recebida, mas a mensagem recebida ter informação nova com relação às mensagens pendentes, então o nodo toma sua parte na disseminação corrente. (4) No caso de nem as mensagens pendentes nem as mensagens recebidas terem informação novas comparadas entre si, então o nodo toma sua parte na disseminação corrente.

Quando um nodo recebe uma confirmação de disseminação (`ReceiveAck`), ele atualiza a variável `AckToReceive`, removendo o identificador do nodo do qual ele recebeu a confirmação. Se `AckToReceive` fica vazio, então uma confirmação é enviada ao nodo em `AckToSend`, o qual é o pai do nodo corrente na árvore de disseminação.

4. Latência de Conectividade

Seja a rede representada por um grafo $G = (V, E)$ onde V é um conjunto de vértices que correspondem aos nodos da rede e E é um conjunto de arestas, cada uma delas representando em canal de comunicação conectando dois vértices em V.

Cada nodo operacional i mantém o grafo G, bem como informação sobre o estado de todos os vértices em V, e todas as arestas em E. O nodo i pode considerar um vértice tanto como *sem-falha* como *inatingível*. Um enlace pode estar tanto *sem-falha*, *silencioso* ou *inatingível*.

A tabela de *timestamps* é utilizada para manter para cada aresta um contador de estados, o qual é inicialmente zero, e é incrementado a cada vez que um *evento de falha* é detectado no enlace correspondente, isto é, o estado do enlace muda de *sem-falha* para *silencioso*. Eventos de recuperação, nos quais um enlace *silencioso* se torna *sem-falha* serão tratados em trabalhos futuros.

De forma a determinar a qual componente conexo o nodo i pertence, ele remove todos os enlaces silenciosos do grafo G, e emprega um algoritmo de conectividade em grafos para determinar quais nodos e enlaces estão inatingíveis. Um componente conexo $G_i = (V_i, E_i)$ é um subgrafo de G do ponto de vista do nodo i onde o vértice v_i pertence a V_i se ele não está inatingível ao nodo i, e aresta e_i pertence a E_i se ela não está inatingível ao nodo i.

Após um novo evento ser detectado num enlace adjacente, um nodo inicia a disseminação de uma mensagem em uma árvore distribuída de busca em largura. Uma *rodada de disseminação*

Algoritmo *Distributed Network Connectivity* Executado pelo Nodo i

|| Module: *BFT-Disseminate (Root, Message)*
DisseminationId ++;
PendingMsg$_{DisseminationId}$.*Message* = *Message*;
PendingMsg$_{DisseminationId}$.*Root* = *Root*; *PendingMsg*$_{DisseminationId}$.*Ack* = 0;
AckToSend$_{DisseminationId}$ = *father in the tree given Root*; /* empty if i is root */
Build Breadth-First Tree based on the updated topology;
AckToReceive$_{DisseminationId}$ = {};
for each node *j* son of node *i* in BFT$_m$
send(*PendingMessage*$_{DisseminationId}$) to node *j*
AckToReceive$_{DisseminationId}$ = *AckToReceive*$_{DisseminationId}$ + {*j*}
end for

|| Module: *Receive (RcvdMsg)*
if exists *PendingMsg*$_m$ *that has newer info compared to RcvdMsg*
then if *RcvdMsg has newer information compared to PendingMsg*$_m$
then update *Status* with received newer info;
for all *m AckToReceive*$_m$ = {}, *AckToSend*$_m$ = {} **end for all**
BFT-Disseminate(*i, merge all pending messages + new event*);
/***else** *drop RecvdMsg* */
endif
else if *RcvdMsg has newer information compared to a PendingMsg*$_m$
then update *Status* with received newer info
for all *m AckToReceive*$_m$ = {}, *AckToSend*$_m$ = {} **end for all**
BFT-Disseminate(*RecvMsg.Root, merge all pending msgs + new event*);
else *BFT-Disseminate*(*RecvdMsg.Root, RecvMsg*)
end if
end if

|| Module: *ReceiveAck (FromNode, AckMsg)*
if exists *PendingMsg*$_m$ *such that*
PendingMsg$_m$.*Root* = *AckMsg.Root* **and** *PendingMsg*$_m$.*Message* = *AckMsg.Message*
then *AckToReceive*$_m$ -= FromNode;
if *AckToReceive*$_m$ == {} **and** *AckToSend*$_m$!= {}
then send(*AckToSend*$_m$, *AckMsg*)
end if
end if

Figura 8: Algoritmo DNC: fase de disseminação.

é definida como o intervalo de tempo no qual todos os nodos em um nível da árvore enviam a mensagem para todos os seus filhos no nível seguinte.

TEOREMA 1. Considere um evento de falha em um enlace, que não particiona um dado componente conexo. Seja d a maior distância mínima entre dois nodos quaisquer que pertençam ao componente, isto é, d é o diâmetro do componente conexo. Considere que ambos os nodos conectados pelo enlace detectam o evento, e que nenhum outro evento ocorre antes que a disseminação deste se complete. Em no máximo $3 * d$ rodadas de disseminação a informação do evento no enlace alcança todos os nodos que pertencem ao componente conexo.

PROVA. Considere que o enlace (i, j) se torna silencioso para ambos os nodos i e nodo j, os quais iniciam árvores de disseminação. Como ambos os nodos pertencem ao mesmo componente conexo, então existe ao menos um nodo k o qual é o primeiro a receber ambas as mensagens de disseminação. A distância do nodo i ao nodo k é, no máximo, d, bem como a distância do nodo j ao nodo k. Então em, no máximo, $2 * d$ rodadas de disseminação um nodo recebe ambas as mensagens de disseminação e inicia uma nova árvore de disseminação com uma mensagem que é a combinação das duas mensagens precedentes. Em, no máximo, mais d rodadas de disseminação a mensagem resultante chega a todos os nodos no componente conectado. Portanto em no máximo $2 * d + d = 3d$ rodadas todos os nodos tomam conhecimento do evento. □

TEOREMA 2. Considere que m eventos de falha em enlace que não particionam um dado componente conexo são detectados antes que a disseminação de qualquer desses eventos complete. Em, no máximo, $2 * m * d + d$ rodadas de disseminação todo nodo conectado recebe informações sobre todos os eventos.

PROVA. Para cada evento de falha em um enlace dois nodos determinam que o enlace se tornou silencioso e iniciam árvores de disseminação. Portanto, para m eventos de falha de enlace $2 * m$ árvores de disseminação são iniciadas. Como estão sendo considerados somente nodos que pertencem ao mesmo componente conexo, então em, no máximo, d rodadas de disseminação pelo menos um nodo que recebe duas mensagens de disseminação inicia a disseminação de uma mensagem que é a combinação das duas mensagens recebidas.

Considerando esta nova árvore de disseminação, em no máximo d rodadas de disseminação a mensagem de disseminação pode chegar ao nodo que tem outra disseminação pendente. Então, para todas as $2 * m$ árvores chegarem a um nodo que combine suas mensagens são necessários, no máximo $2 * m * d$ rodadas de disseminação.

Finalmente, em no máximo mais d rodadas de disseminação a mensagem resultante alcança todos os nodos no componente conexo. Portanto em no máximo $2 * m * d + d$ rodadas todos os nodos tomam conhecimento sobre todos os eventos. □

TEOREMA 3. Considere um evento de falha em um enlace o qual particiona um dado componente conexo em dois componentes conexos. Sejam d_1 e d_2 os diâmetros dos componentes conexos resultantes. Considere que a informação sobre todos os eventos precedentes nesses componentes já foram completamente disseminadas. Considere também que nenhum outro evento ocorre nesses componentes. Em, no máximo, d_1 e d_2 rodadas de disseminação, respectivamente, todo nodo em cada componente conexo recebe a informação sobre o particionamento.

PROVA. Considere que o enlace (i, j) se torna silencioso para ambos os nodos i e j, os quais iniciam árvores de disseminação. Considerando que o enlace (i, j) particiona o sistema em dois componentes conexos, estando o nodo i em um desses componentes e o nodo j no outro. Como d_1 e d_2 são os diâmetros dos componentes conexos resultantes, as árvores montadas pelos nodos i e nodo j levam, no máximo, d_1 e d_2 rodadas de disseminação para chegar em todos os nodos conectados no respectivo componente. □

TEOREMA 4: Considere um evento de falha em um enlace que particiona um dado componente conexo em dois componentes conexos. Considere também que t disseminações estão pendentes em um dos componentes, usando a topologia do sistema como estava antes do particionamento. Seja d o diâmetro do componente conexo. Considere também que nenhum outro evento ocorre nesse componente. Em, no máximo, $t * d + d$ rodadas de disseminação todos os nodos no componente conexo tomam conhecimento sobre o particionamento e todos os evento que estavam sendo disseminados antes do particionamento.

PROVA. Considere que o enlace (i, j) se torna silencioso para o nodo i, o qual inicia uma árvore de disseminação. Considerando que o enlace (i, j) particiona o sistema em dois componentes conexos, estando o nodo i em um desses componentes e o nodo j no outro. Como d é o diâmetro do componente do nodo i, a árvore iniciada pelo nodo i leva no máximo d rodadas de disseminação para alcançar o nodo k, o qual tem uma das disseminações pendentes naquele componente. O nodo k toma conhecimento sobre o particionamento e inicia a disseminação de uma mensagem que é a combinação de ambas as outras.

Considerando esta nova árvore, em no máximo d rodadas a mensagem de disseminação chega ao nodo que tem outra disseminação pendente. Então, para todas as t árvores alcançarem um nodo que combine suas mensagens, são necessários, no máximo, $t * d$ rodadas de disseminação.

Finalmente, em, no máximo, mais d rodadas de disseminação a mensagem resultante alcança todos os nodos no componente conexo. Portanto em, no máximo, $t * d + d$ rodadas todos os nodos são informados sobre todos os eventos. □

5. Resultados Experimentais

Nesta seção, são apresentados resultados experimentais obtidos através de simulação. Os resultados são comparados com resultados de duas outras abordagens: um algoritmo paralelo baseado em inundação [11] e um algoritmo sequencial baseado no Agente Chinês [12]. Várias topologias de rede são consideradas: inicialmente são apresentados resultados para o grafo de exemplo mostrado na figura 2. Em seguida, são apresentados resultados para hipercubos com 16, 64 e 128 nodos; depois o algoritmo foi simulado num grafo randômico de 50 nodos, o grafo $D_{1,2}$ com 9 nodos, e um subconjunto da toplogia da RNP (*Rede Nacional de Pesquisa*).

As simulações foram feitas usando SMPL *SiMulation Programming Language* [10], uma biblioteca de simulação de eventos discretos. Para cada experimento de simulação foi escalonado um evento de falha num enlace de comunicação. São utilizados intervalos de teste de 30 unidades de tempo e intervalos de disseminação de 1 unidade de tempo entre nodos conectados por um enlace.

5.1. Uma Topologia Exemplo

Para a topologia de exemplo de 7 nodos ilustrada na figura 2, resultados da execução do algoritmo após a ocorrência de um evento de falha no enlace 1–3, como descrito na seção precedente, são mostrados na tabela 1.

	Total deMensagens	MensagensRedundantes	Latência
DNC	12	0	8
Inundação	28	16	7
Agente Chinês	7	1	7

Tabela 1: Resultados para uma topologia exemplo.

5.2. Grafo $D_{1,2}$

Para o grafo da topologia $D_{1,2}$ [8], foi escalonado um evento de falha para o enlace 6–8. O nodo 6 inicia a disseminação montando uma árvore de busca em largura de quatro níveis com raiz em si mesmo. Desta forma, decorrem 2 unidades de tempo para a disseminação alcançar o nodo 8. Naquele instante, o nodo 8 inicia outra árvore de disseminação sobre o evento no enlace 6–8 mais a informação sobre o evento de falha no enlace 8–6. Para esta disseminação, é iniciada outra árvore de quatro níveis. Então decorrem três unidades de tempo para a informação alcançar as folhas desta árvore, e mais três unidades de tempo para as confirmações chegarem ao nodo 8, na raiz. Resultados comparativos com os outros algoritmos mencionados são mostrados na tabela 2.

	Total de Mensagens	Mensagens Redundantes	Latência
DNC	16	0	9
Inundação	52	36	7
Agente Chinês	13	5	13

Tabela 2: Resultados para $D_{1,2}$ de 9 nodos.

5.3. Hipercubos

Abaixo são descritos os resultados obtidos para hipercubos com 16, 64, e 128 nodos. Para o hipercubo de 16 nodos é escalonado um evento de falha no enlace 5-7. O nodo 5 detecta o evento e inicia a disseminação. Uma árvore de cinco níveis é montada, com o nodo 7 no quarto nível. Três unidades de tempo após o início, o nodo 7 é informado sobre o evento no enlace 5–7 e inicia a disseminação do evento no enlace 7–5. Esta disseminação usa outra árvore de cinco níveis, de tal forma que quatro unidades de tempo após o nodo 7 iniciar a disseminação, a informação chega à base da árvore. Quatro unidades de tempo após o que a última mensagem de confirmação chega ao topo. Resultados para as outras abordagens são também mostrados na tabela 3.

	Total de Mensagens	Mensagens Redundantes	Latência
DNC (16 nodos)	30	0	12
Inundação (16 nodos)	94	64	9
Agente Chinês (16 nodos)	28	14	28
DNC (64 nodos)	126	0	24
Inundação (64 nodos)	478	352	15
Agente Chinês (64 nodos)	156	93	156
DNC (128 nodos)	254	0	36
Inundação (128 nodos)	990	736	23
Agente Chinês (128 nodos)	348	221	348

Tabela 3: Resultados para os hipercubos de 16, 64 e 128 nodos.

Considerando o hipercubo de 64 nodos com um evento de falha detectado no enlace 5–7 pelo nodo 7, a disseminação é realizada com 126 mensagens em 24 unidades de tempo. Considerando a topologia de hipercubo de 128 nodos com um evento de falha no enlace 13–21, a disseminação inicia no nodo 13 e usa 254 mensagens e 36 unidades de tempo para completar.

5.4. Um Grafo Randômico

Para esta simulação um grafo randômico foi criado com uma probabilidade de 10nodos é igual a 50. O grafo, contudo, consiste de um componente conexo.

Para este grafo, um evento no enlace 30–31 é escalonado e a disseminação inicia no nodo 31. Este nodo emprega uma árvore de busca em largura de cinco níveis, com o nodo 30 no quarto

nível. Então, três unidades de tempo após o início, a disseminação alcança o nodo 30. Neste instante, o nodo 30 monta outra árvore de cinco níveis e inicia a disseminação da informação do evento de falha no enlace 30–31 junto com a informação recebida sobre o evento no enlace 31–30. Quatro unidades de tempo após esta disseminação iniciar, ela chega aos nodos folha, e outras quatro unidades de tempo após isso são necessárias para as mensagens de confirmação chegarem ao nodo 30, na raiz. Resultados comparativos são mostrados na tabela 4.

	Total of Mensagens	Mensagens Redundantes	Latência
DNC	143	0	14
Inundação	418	320	8
Agente Chinês	109	60	109

Tabela 4: Resultados para um grafo randômico.

5.5. Um Subconjunto da *RNP*

A topologia da *RNP* empregada para as simulações foi aquela disponível em fevereiro de 2001. Tratava-se de uma topologia de 28 nodos e um evento de falha foi escalonado para o enlace São Paulo–Brasília.

Uma vez detectada a falha, o nodo Brasília inicia a fase de disseminação empregando uma árvore de busca em largura de quatro níveis. Dois níveis abaixo da raiz da árvore está o nodo São Paulo, que recebe a informação duas unidades de tempo após o início da disseminação. Em seguida, o nodo São Paulo inicia outra disseminação para informar o evento de falha no enlace São Paulo–Brasília e o faz construindo uma árvore de busca em largura de quatro níveis com raiz em si mesmo. Três unidades de tempo após a segunda disseminação ser iniciada ela alcança as folhas. Três unidades de tempo após o que, as mensagens de confirmação chegam ao topo da árvore, completando a disseminação. Resultados comparativos são mostrados na tabela 5.

	Total deMensagens	Mensagens Redundantes	Latência
DNC	54	0	9
Inundação	94	40	6
Agente Chinês	55	27	55

Tabela 5: Resultados para um subconjunto da *RNP* brasileira.

5.6. Considerações Finais

O algoritmo de inundação sempre completa a disseminação no intervalo de tempo mais curto possível. Emprega, entretanto, um grande número de mensagens, com muitas mensagens redundantes. O número de mensagens empregado pelo Agente Chinês é sempre muito menor que o número de mensagens empregado pela Inundação. O Agente Chinês apresenta também o menor impacto possível na rede por haver no máximo uma mensagem sendo disseminada a um dado instante de tempo. O número de mensagens empregado pelo DNC é similar ao número empregado pelo Agente Chinês, enquanto que a latência do DNC é similar à da Inundação. Dado o conjunto de experimentos acima pode-se concluir que o algoritmo DNC tem, comparativamente, baixa latência, apresentando um baixo impacto na rede em número de mensagens.

6. Conclusão

O algoritmo *Distributed Network Connectivity* descrito neste artigo permite a qualquer nodo de uma rede de topologia arbitrária calcular quais porções da rede são atingíveis, e quais porções

são inatingíveis. Enlaces são testados continuamente, durante intervalos de teste, disseminando informações sobre enlaces silenciosos através de uma árvore distribuída de busca em largura. A qualquer tempo, qualquer nodo operacional pode calcular a conectividade da rede. Limites do pior caso da latência do algoritmo são provados. Resultados de simulação da disseminação de um evento de falha são apresentados para uma topologia de exemplo; o grafo $D_{1,2}$; hipercubos de 16, 64, e 128-nodos; um grafo randômico, e um subconjunto da RNP brasileira. Os resultados foram comparados àqueles de dois outros algoritmos.

Trabalhos futuros incluem permitir que o algoritmo trabalhe na presença de eventos de recuperação. Uma ferramenta prática baseada no protocolo de gerenciamento da Internet, SNMP (*Simple Network Management Protocol*) [2] está sendo desenvolvida. Esta ferramenta apresenta uma interface Web que permite a qualquer usuário obter a informação sobre a conectividade da rede a partir de qualquer nodo da mesma rede. Outros trabalhos futuros incluem desenvolver estratégias efetivas de teste para enlaces WAN, específicos para as tecnologias de comunicação utilizadas.

Referências

[1] W. Stallings, *Snmp, Snmpv2, Snmpv3 and Rmon 1 and 2*, Addison-Wesley, 1999.

[2] D. Harrington, R. Presuhn and B. Wijnem, "An Architecture for Describing SNMP Management Frameworks," *RFC 2571*, IETF, May 1999.

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

[4] E.P. Duarte Jr., G. Mansfield, T. Nanya, and S. Noguchi, "Non-Broadcast Network Fault Monitoring Based on System-Level Diagnosis," *Proc. IEEE/IFIP IM'97*, pp.597-609, San Diego, May 1997.

[5] E.P. Duarte Jr., "Um algoritmo para Diagnóstico de Redes de Topologia Arbitrária," *in portuguese*, In Proceedings of the *1st SBC Workshop on and Fault Tolerance*, SBC-WTF'1, pp. 50-55, Porto Alegre, Brazil, 1998.

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

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

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

[9] 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.

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

[11] E.P. Duarte Jr., and G.O. Mattos, "Diagnóstico em Redes de Topologia Arbitrária: Um Algoritmo Baseado em Inundação de Mensagens", *in portuguese*, In Proceedings of the *2nd SBC Workshop on Test and Fault Tolerance*, SBC-WTF'2, pp. 82-87, Curitiba, Brazil, 2000.

[12] E.P. Duarte Jr., and J.M.A.P. Cestari, "O Agente Chinês para Diagnóstico de Redes de Topologia Arbitrária" *in portuguese*, In Proceedings of the *2nd SBC Workshop on Test and Fault Tolerance*, SBC-WTF'2, pp. 88-93, Curitiba, Brazil, 2000.