JONATAN SCHROEDER

ROTEAMENTO DINÂMICO TOLERANTE A FALHAS BASEADO EM AVALIAÇÃO DE FLUXO MÁXIMO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2006

JONATAN SCHROEDER

ROTEAMENTO DINÂMICO TOLERANTE A FALHAS BASEADO EM AVALIAÇÃO DE FLUXO MÁXIMO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2006

Agradecimentos

Agradeço a Deus por ter conduzido todo este trabalho. Sem Ele não haveria nem mesmo uma página. Agradeço ao professor Elias Procopio Duarte Jr. pela orientação deste trabalho, com todo o esforço que empreendeu para que a conclusão deste projeto. Agradeço a meus pais e minha família pelo apoio e suporte que me deram, sem pedir nada em troca. Agradeço ao professor Jaime Cohen por ter sido o responsável pelo gatilho inicial deste trabalho, lançando a idéia do roteamento utilizando conectividade. Agradeço aos colegas e amigos que sempre me incentivaram a continuar este trabalho. Agradeço à UFPR e à CAPES pelo apoio institucional e financeiro que deram a este trabalho.

Sumário

Lista de Figuras

Glossário de Siglas

ACK	*Acknowledgement*
AS	*Autonomous System*
ATM	*Asynchronous Transfer Mode*
BGP	*Border Gateway Protocol*
BSD	*Berkeley Software Distribution*
CIDR	*Classless InterDomain Routing*
EGP	*Exterior Gateway Protocol*
EIGRP	*Enhanced* IGRP
FDDI	*Fiber Distributed Data Interface*
IAB	*Internet Architecture Board*
IETF	*Internet Engineering Task Force*
IGP	*Interior Gateway Protocol*
IGRP	*Interior Gateway Routing Protocol*
IP	*Internet Protocol*
IPv6	IP *version 6*
IS-IS	*Intermediate System to Intermediate System*
J2SE	*Java 2 Standard Edition*
MIB	*Management Information Base*
OSI	*Open Systems Interconnection*
OSPF	*Open Shortest Path First*
QoS	*Quality of Service*
RFC	*Request For Comments*
RIP	*Routing Information Protocol*
RIPng	RIP *next generation*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

Resumo

A Internet está sujeita a alterações em sua topologia. Essas alterações são resultado tanto de inclusões e remoções, como de falhas e recuperações de nós e enlaces. Como resultado dessas alterações, as tabelas de rotas dos protocolos de roteamento ficam incorretas. Esses protocolos possuem uma latência, isto é, um intervalo de tempo para atualizar suas tabelas de rotas em toda a rede com informações atuais relativas à topologia. Durante essa latência, pacotes enviados na rede podem ser perdidos, bem como conexões podem ser rompidas. Este trabalho propõe uma estratégia de roteamento dinâmico, permitindo que roteadores intermediários, que podem possuir informações mais recentes de alterações na topologia, interfiram na escolha do caminho utilizado. A estratégia de roteamento proposta baseia-se na escolha de arestas para roteamento utilizando cálculo de fluxo máximo em grafos, aumentando o número de caminhos disjuntos, o que valoriza a redundância de caminhos e, por conseqüência, ampliam a possibilidade de utilização de desvios ou caminhos alternativos. Critérios de distância da rota são utilizados como critérios secundários. A abordagem de roteamento é formalmente especificada. São apresentadas provas de correção do algoritmo, número e tamanho das mensagens de atualização, complexidade e latência de convergência do roteamento. Resultados experimentais obtidos através de simulação em redes com topologias similares à da Internet são apresentados.

Abstract

The Internet topology is dynamic, i.e. it changes with time. These changes are a result of the failure and recovery of both network nodes and links, as well as the inclusion and removal of nodes and links. As a result of these changes, route tables used by routing protocols must be updated. These protocols have a latency, that is, a time interval required to update their routing tables throughout the network with current information about the topology. During this latency, packets sent through the network are lost, as well as connections are broken. This work proposes a dynamic routing strategy, in which intermediate routers, having more recent information about topology changes, are able to switch the path employed. The proposed routing strategy chooses network edges for routing based on maximum flow evaluation, in order to increase the number of disjoint paths, enhancing the path redundancy, and so extending the possibility of using detours, or alternative paths. Route distance is employed as a secondary criterion. The routing approach is formally specified. Correctness proofs for the algorithm are presented, as well as proofs for the number and size of messages required, the complexity and the convergence latency of the algorithm. Experimental results obtained from simulation of the algorithm run on networks with Internet-like topologies are also presented.

Capítulo 1

Introdução

As topologias das redes de computadores, especialmente a da Internet, se alteram com o tempo, em maior ou menor grau, na medida em que ocorre o acréscimo e a remoção de nós (pontos de rede) e enlaces (ligações entre nós), além de falhas e recuperações nos nós e enlaces.

Os protocolos de roteamento da Internet, que definem os caminhos utilizados para a comunicação entre os nós, baseiam suas decisões em tabelas locais a cada nó. Essas tabelas possuem informações a respeito da topologia da rede, além de parâmetros que permitem a escolha do caminho a ser adotado para comunicação.

À medida em que eventos de alteração de topologia ocorrem, os protocolos de roteamento ficam, por um certo tempo, com uma informação de topologia desatualizada. Esse tempo de convergência é necessário para que o protocolo atualize suas tabelas de rotas em todos os roteadores. Esse período de tempo é conhecido como *latência de convergência* do protocolo [1]. A latência média do protocolo BGP (*Border Gateway Protocol*), um dos mais utilizados na Internet atualmente [2], é de 3 minutos, sendo que já foram observados períodos de latência de até 15 minutos [1].

Durante a latência de convergência, pacotes enviados pela rede podem ser potencialmente perdidos, bem como conexões podem ser rompidas. Diversas abordagens têm sido propostas para evitar a perda de pacotes e conexões durante a latência de convergência dos protocolos de roteamento. Chandrashekar et.al. [3] propõem um mecanismo para análise

das dependências de caminhos, de forma a reduzir a exploração de novos caminhos e, por conseqüência, reduzir a latência de convergência. Pei et.al. [4] propõem uma série de asserções a serem aplicadas nas redes, com o propósito de comparar caminhos similares e eliminar caminhos inviáveis. Wang et. al. [5] citam uma estratégia baseada em um mecanismo escalável, denominado FRTR (*Fast Routing Table Recovery*), para detecção e recuperação de inconsistências nas tabelas de roteamento do protocolo BGP.

Uma abordagem para aumentar a tolerância a falhas dos caminhos escolhidos, proposta em [6, 7], baseia-se na identificação de nós altamente conexos na rede, com o intuito de encontrar caminhos que passem por esses nós. Propõe-se que, em caso de ocorrência de falhas na conexão, seja encontrado um desvio que passa pelo nó altamente conexo de forma a chegar ao destino. Há uma ênfase na conectividade individual dos nós altamente conexos, sem que haja uma preocupação com a conectividade dos demais nós do caminho escolhido para a comunicação.

Neste trabalho é proposta uma estratégia de roteamento dinâmico, permitindo que roteadores intermediários, que podem possuir informações mais recentes de alterações na topologia, interfiram na escolha do caminho utilizado. A proposta é detalhada na próxima seção.

1.1 Roteamento Dinâmico e Tolerante a Falhas

Neste trabalho é proposta uma abordagem de roteamento que faz a seleção de rotas tendo como um dos principais objetivos a geração de uma rota robusta, no sentido de que, no caso da ocorrência de falhas em parte do caminho (nós ou enlaces), é possível encontrar outro caminho (ou *desvio*) que parte do ponto em que a falha é conhecida em direção ao destino da mensagem sendo roteada. A avaliação da robustez do caminho é feita com base no tamanho desse desvio em relação à distância dos nós de origem e destino na rede. A robustez de uma rota utilizada é maior quanto mais próximo for o número total de nós visitados do caminho mínimo entre os nós na rede.

Além da robustez dos caminhos escolhidos, outra característica importante é que o roteamento proposto neste trabalho é dinâmico. Em um roteamento dinâmico, cada nó

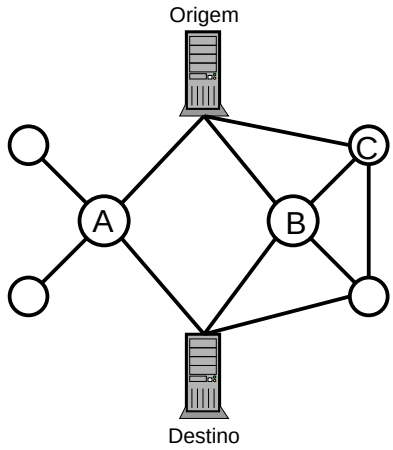

Figura 1.1: Exemplo de vantagem do roteamento proposto neste trabalho.

que recebe a mensagem a ser roteada escolhe apenas a próxima aresta da rota, permitindo que roteadores intermediários da rota possam escolher uma rota melhor que as conhecidas pelo nó de origem da mensagem. Esse roteamento explora o fato de que nós mais próximos a um evento de alteração de estado de nó ou enlace recebem a informação do evento antes dos demais nós da rede. Este conceito pode ser estendido também a informações de congestionamento, caso em que uma aresta pode ser considerada temporariamente falha.

A precisão do algoritmo proposto está diretamente relacionada com a correção das informações de topologia disponíveis em cada nó. À medida em que a topologia da rede é alterada, todos os nós da rede devem atualizar suas tabelas de acordo com a alteração da topologia. Porém, visto que um dos focos deste trabalho é a tolerância a falhas, o algoritmo funciona mesmo que a tabela não esteja atualizada, ou possua uma posição incorreta da topologia da rede.

A figura 1.1 ilustra a utilização do roteamento dinâmico. Suponha que exista uma falha na aresta que liga B ao destino, e que o nó de origem não desconheça essa informação. O nó de origem da mensagem, ao enviar essa mensagem ao destino através do nó B, não sabe que a aresta que liga B ao destino está falha. O nó B, ao receber a mensagem, já tem conhecimento da falha, e, portanto, pode escolher um caminho alternativo. Em suma, a mensagem chega ao destino, mesmo que o nó de origem não tenha conhecimento da falha ocorrida.

Este trabalho possui como objetivo a escolha de arestas tais que, em caso de falha,

não seja necessário voltar à origem do roteamento para escolher um desvio. Tal objetivo pode ser alcançado através da avaliação de caminhos disjuntos. Dois caminhos entre um par de nós da rede são disjuntos se não há nenhuma aresta em comum entre os caminhos.

O cálculo do número de caminhos disjuntos pode ser obtido utilizando algoritmos para o cálculo de fluxo máximo [8, 9]. Um fluxo em um grafo é a atribuição de valores (fluxos) às arestas de um grafo direcionado em que cada aresta possui uma capacidade, de tal forma que os valores não excedam a capacidade das arestas. O fluxo das arestas que chegam a um nó deve ser igual ao fluxo das arestas que saem do mesmo, exceto pela origem e pelo destino. Um fluxo é máximo quando o valor total do fluxo que sai da origem é máximo. Fluxos são normalmente utilizados para simular tráfego em sistemas rodoviários, fluxo de fluidos em encanamentos, correntes em circuitos elétricos, entre outros. O cálculo do fluxo máximo pode ser realizado com algoritmos como o de Ford-Fulkerson [8].

A figura 1.1 também pode ser utilizada para ilustrar o cálculo do fluxo máximo. Na figura, a origem do roteamento pode escolher entre 3 vizinhos para o roteamento, A, B e C. Entre A e o destino há apenas um caminho possível, porém entre B e o destino há 2 caminhos disjuntos que podem ser utilizados, o que também acontece com C. Desta forma, o roteamento proposto neste trabalho dará preferência aos nós B e C em relação ao nó A, visto que há uma possibilidade maior da utilização de desvios através destes nós.

A distância dos caminhos utilizados também é empregada neste trabalho para a escolha de arestas para roteamento, porém como critério secundário. Utilizando o mesmo exemplo descrito anteriormente, na figura 1.1, o roteamento descrito neste trabalho dará preferência ao nó B em relação ao nó C para realizar o roteamento do exemplo, tendo em vista que B possui um caminho mais curto que os caminhos disponíveis através de C.

O modelo de falhas considerado neste trabalho é o das falhas *crash*, e o sistema é considerado parcialmente síncrono, ou seja, existe um limite de tempo finito e não necessariamente conhecido para o atraso na comunicação entre dois nós quaisquer, possibilitando a diferenciação entre uma falha de nó ou enlace e um atraso na comunicação.

A abordagem do roteamento proposto é formalmente especificada neste trabalho. São apresentadas provas de correção do algoritmo, juntamente com teoremas relativos ao

número e tamanho das mensagens de atualização de topologia, complexidade do algoritmo de roteamento e latência de convergência. Este trabalho também apresenta uma implementação do roteamento descrito, o seu uso para experimentação e geração de resultados, bem como uma interface gráfica para visualização do funcionamento do mesmo. A implementação e os resultados experimentais gerados através da implementação, baseados em um ambiente de simulação em redes com topologia similar à da Internet são descritos no capítulo 4. Esses resultados compreendem avaliações do funcionamento e correção do algoritmo, assim como o comportamento do algoritmo de acordo com as parametrizações do mesmo.

1.2 Organização deste Trabalho

Este trabalho está organizado como segue. No capítulo 2 são descritos conceitos preliminares utilizados no trabalho, como a arquitetura de roteamento da Internet, incluindo os algoritmos e protocolos de roteamento, e trabalhos relacionados, incluindo as propostas de roteamento específicas para redes sem fio, roteamento tolerante a falhas e roteamento baseado em QoS (*Quality of Service*). No capítulo 3 é detalhada a especificação do algoritmo proposto neste trabalho, juntamente com as provas de correção do mesmo. O capítulo 4 descreve a implementação desenvolvida para a estratégia de roteamento proposta, bem como resultados obtidos por simulação do algoritmo. Por fim, o capítulo 5 apresenta as conclusões.

Capítulo 2

Roteamento em Redes de Computadores

Este capítulo tem como propósito apresentar trabalhos relacionados de roteamento em redes de computadores, além de introduzir conceitos básicos relativos ao roteamento em redes de computadores, especialmente na Internet.

O restante do capítulo está organizado como segue. A seção 2.1 descreve a arquitetura e funcionamento dos algoritmos e protocolos de roteamento na Internet. A seção 2.2.1 descreve algumas políticas para roteamento em redes sem fio *ad-hoc*. A seção 2.2.2 descreve algumas estratégias descritas para o roteamento baseado em QoS. A seção 2.2.3 descreve algoritmos utilizados para roteamento tolerante a falhas.

2.1 Arquitetura de Roteamento na Internet

A Internet pode ser vista como um conjunto de redes de diferentes tamanhos, formatos, arquiteturas e topologias. Cada uma dessas redes é constituída de um ou mais roteadores, que interligam arbitrariamente essas redes entre si utilizando protocolos em comum, como o protocolo IP (*Internet Protocol*). Desta forma, a estrutura da rede possibilita que, mesmo com arquiteturas diferentes, os pontos de rede possam estabelecer conexões e transmitir dados entre si.

As diversas redes e roteadores da Internet estão organizadas em entidades chamadas

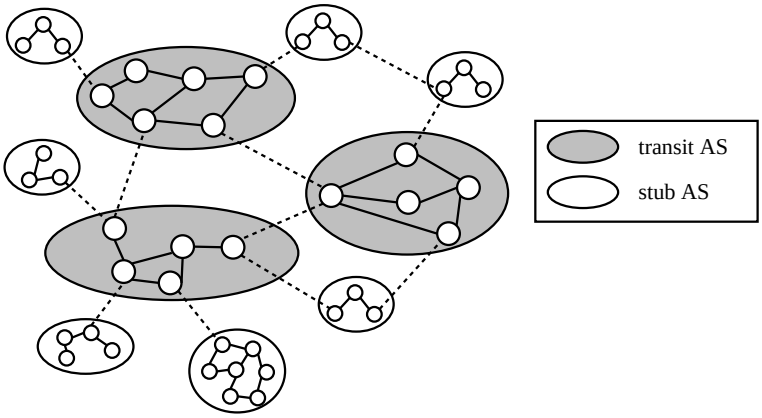

Figura 2.1: Interconexões entre *transit AS's* e *stub AS's*.

Sistemas Autônomos (AS - *Autonomous Systems*) [10]. Um sistema autônomo é um conjunto de roteadores sob uma única administração técnica. Cada AS é responsável por sua própria política de roteamento, especificando quais rotas são anunciadas a outros AS's e atribuindo graus de preferência para as rotas.

Nas comunicações estabelecidas entre nós de um mesmo AS, é utilizado um protocolo de roteamento interno (IGP - *Interior Gateway Protocol*). Nós em AS's distintos utilizam roteamento denominado externo. Cada um desses tipos de roteamento possui características distintas. O roteamento interno tem à sua disposição a topologia completa do AS, permitindo uma seleção de acordo com os objetivos da política de roteamento do AS. O roteamento externo, por sua vez, possui uma complexidade maior, visto que envolve vários AS's que, em conjunto, são utilizados para seleção dos caminhos.

Os roteadores de um AS que fazem comunicação com roteadores de outros AS's são conhecidos como roteadores de borda. Esses roteadores são responsáveis por anunciar rotas disponíveis para outros AS's.

Os AS's são classificados como *transit AS* e *stub AS*. Os *transit AS's* formam o *backbone* (espinha dorsal) da Internet e servem para interconectar *stub AS's* de forma eficiente. Um *stub AS* deve se conectar a um ou mais *transit AS*, podendo ocasionalmente conectar-se a outro *stub AS*. A figura 2.1 apresenta um exemplo da interligação entre esses tipos de AS's. Nessa figura, os círculos pequenos representam roteadores, que estão organizados em AS's. Os *transit AS's* estão destacados e centralizam a comunicação entre os *stub AS's*.

Os protocolos de roteamento implementam algoritmos de roteamento. Alguns dos mais conhecidos algoritmos de roteamento são apresentados na seção 2.1.1. Alguns dos protocolos mais utilizados na Internet, tanto internos quanto externos, são apresentados na seção 2.1.2.

2.1.1 Algoritmos de Roteamento da Internet

O *algoritmo de roteamento* é a parte do software da camada de rede responsável pela decisão sobre a linha de saída a ser usada na transmissão do pacote de entrada. Existem determinadas propriedades que são desejáveis em um algoritmo de roteamento: correção, simplicidade, estabilidade, eqüidade, otimização e robustez [11]. A correção e a simplicidade são auto-explicativos; a estabilidade refere-se à capacidade de convergir a um equilíbrio, independente do tempo em que o algoritmo é executado; a eqüidade corresponde ao tratamento igual de todos pares origem-destino que desejam comunicar-se pela rede, e a otimização refere-se à utilização da banda de tal forma que a eficiência global seja máxima (sem desprezar, porém, a eqüidade).

Dentre as propriedades citadas acima para um algoritmo de roteamento, nesse trabalho é dada especial importância à robustez, propriedade pela qual o algoritmo eficientemente atualiza suas tabelas de rotas à medida em que atualizações de topologia ocorrem. Durante o funcionamento de um algoritmo de roteamento, falhas de hardware e software podem ocorrer, o número de hosts, roteadores e linhas poderá aumentar e diminuir repetidamente. Tais alterações provocam mudanças freqüentes na topologia. O algoritmo de roteamento deve ser capaz de aceitar as alterações na topologia e no tráfego sem a necessidade de interromper tarefas dos hosts, bem como de reinicializar a rede sempre que algum roteador apresentar falha.

Os algoritmos de roteamento podem ser agrupados em duas classes principais: adaptativos e não-adaptativos. Os *algoritmos não-adaptativos* (*nonadaptive algorithms*) não baseiam suas decisões de roteamento em medidas ou estimativas do tráfego e da topologia atuais. Na verdade, a escolha da rota a ser utilizada é previamente calculada off-line, sendo transferida para os roteadores quando a rede é inicializada. Os *algoritmos adap-*

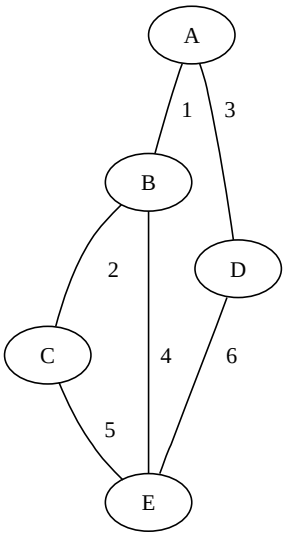

Figura 2.2: Grafo utilizado para apresentação do algoritmo de Bellman-Ford.

tativos (*adaptive algorithms*), em comparação, mudam suas decisões de roteamento para refletir mudanças na topologia e, normalmente, no tráfego também [11].

A seguir são apresentados alguns dos algoritmos mais utilizados para roteamento na Internet, com destaque aos algoritmos de Bellman-Ford e de Dijkstra.

2.1.1.1 Roteamento pelo Algoritmo de Vetor-Distância (Bellman-Ford)

O algoritmo de Bellman-Ford, também conhecido por algoritmo de vetor-distância, foi originalmente descrito em [12]. Os protocolos baseados em vetor-distância têm sido utilizados desde as primeiras redes, como a Arpanet [13] e a Cyclades [14]. Esses protocolos foram usados como base para o desenvolvimento de protocolos atualmente empregados na Internet, como o RIP (*Routing Information Protocol*) [15].

O algoritmo é descrito através de um exemplo, com o auxílio do grafo apresentado na figura 2.2. Assume-se que cada nó possui pelo menos a informação de sua própria identidade e de quais são os enlaces (arestas) adjacentes ao nó. Assume-se também que esse conhecimento é local, ou seja, um nó só possui as informações referentes ao próprio nó e as informações que foram repassadas ao nó por outros nós [16].

Cada nó possui uma tabela contendo os nós com os quais pode se comunicar, jun-

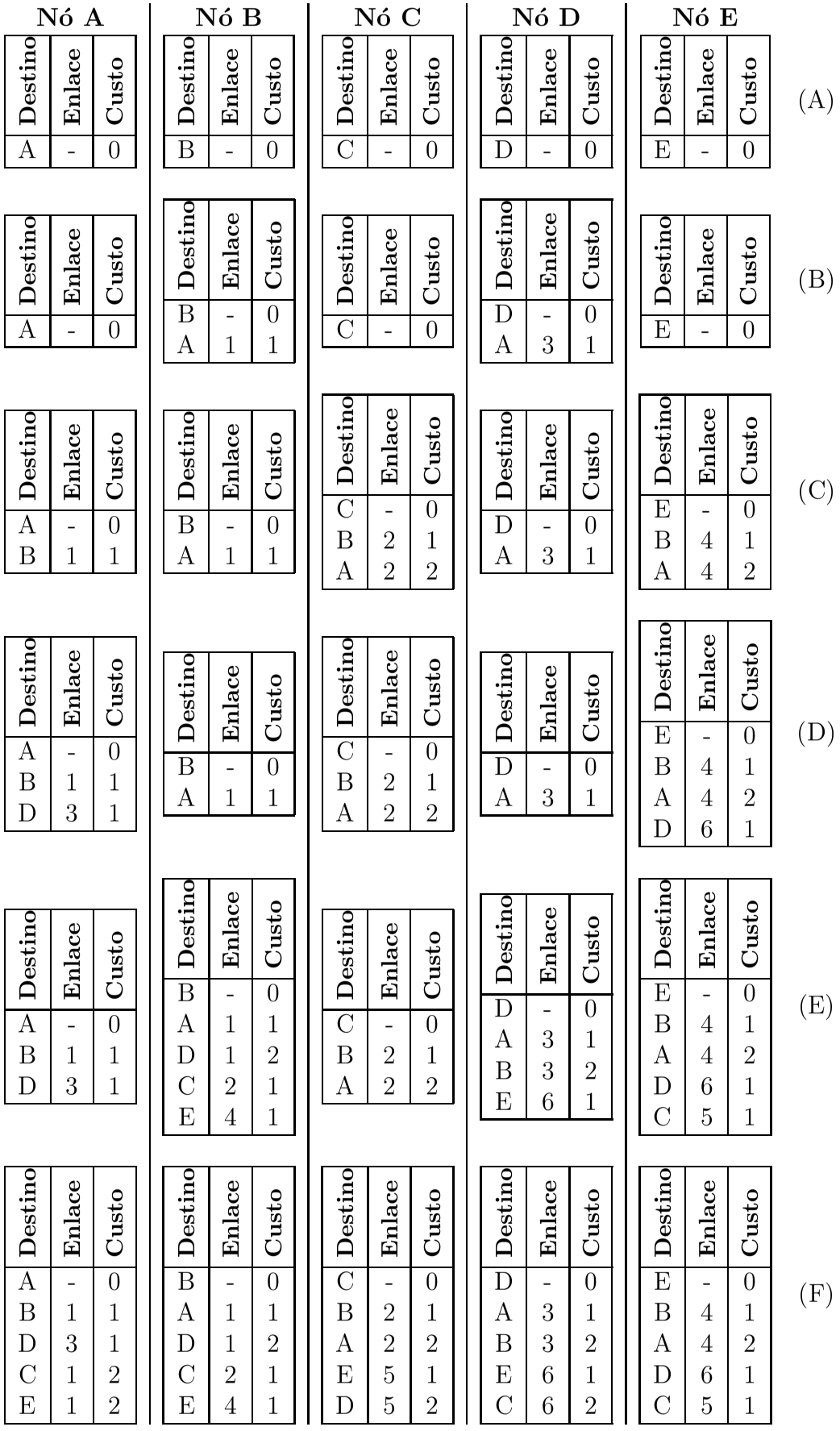

Figura 2.3: Tabelas de vetor-distância do algoritmo de Bellman-Ford.

tamente com a distância até esses nós e qual o enlace inicial utilizado para realizar a comunicação. A figura 2.3 apresenta um exemplo que mostra o conteúdo das tabelas em diferentes momentos, e que são descritos na seqüência. A posição inicial das tabelas é apresentada em (A). Cada coluna apresenta a tabela disponível em cada nó da rede.

Inicialmente, cada nó envia uma mensagem de *broadcast* para todos os seus vizinhos, repassando sua tabela de roteamento. Parte-se do nó A, que envia uma mensagem para B e D. Cada um dos nós inclui as informações repassadas pelo nó A em sua tabela de roteamento, indicando de que enlace essa informação foi recebida. Como a informação atravessou uma aresta, e o custao das arestas é assumido ser igual a um, a distância é acrescida de uma unidade. O resultado é ilustrado em (B).

Uma vez que B recebeu uma mensagem, ele enviará para seus vizinhos A, C e E sua tabela atualizada de roteamento. Esses nós, por sua vez, atualizam suas próprias tabelas com as informações recebidas. O nó A, porém, que atualizaria sua tabela com a entrada A/1/2, já possui um caminho com distância menor para A, e por isto não atualiza sua tabela de roteamento. O resultado é ilustrado em (C).

O nó D, que também recebe a mensagem de A, envia sua tabela atualizada para seus vizinhos A e E. O nó A atualiza sua tabela incluindo o nó D. Já o nó E, além de incluir o nó D, tem a possibilidade de atualizar sua tabela em relação ao nó A. Porém a distância existente na tabela e a distância do novo nó é igual. Assume-se, nesse caso, que o registro é mantido. O resultado após essa atualização é ilustrado em (D).

Os nós A, C e E, que receberam novas informações de roteamento, vão repassar essas informações para seus vizinhos, resultando nas tabelas disponíveis em (E). Em seguida, os nós B, D e E, que foram atualizados, repassam suas informações para seus vizinhos, resultando nas tabelas mostradas em (F). Após essa última alteração, os nós atualizados repassam informações aos vizinhos, mas que não afetam mais as tabelas correspondentes. Nesse ponto é assumido que o algoritmo convergiu, e os nós descobriram as informações que precisavam para o seu roteamento.

O roteamento nas tabelas vetor-distância é feito utilizando as tabelas ilustradas no exemplo acima. Se um nó precisa enviar um pacote para outro, ele consulta sua tabela

para identificar por qual enlace deve enviar o pacote. O nó que recebe a informação repassa para o enlace que encontra-se em sua tabela de vetor-distância, e assim por diante até que o pacote chegue ao seu destino.

Vamos supor, no mesmo exemplo, que, após a convergência do algoritmo, o enlace 1, entre os nós A e B, falhe. Nesse caso, há uma nova execução do algoritmo, cujas tabelas encontram-se na figura 2.4. A execução do algoritmo é discriminada a seguir. Inicialmente, os nós adjacentes à aresta falha atualizam suas tabelas, atribuindo custo ∞ aos nós alcançáveis pela aresta falha. O resultado é mostrado em (A).

Em seguida, os nós A e B enviam suas tabelas atualizadas para os seus respectivos vizinhos. O nó D, recebendo os novos dados de A, verifica que todos os dados recebidos possuem distância superior às de sua própria tabela. Porém, o nó D verifica que, como as informações são provenientes do enlace 3, correspondem a atualizações de topologia vindos desse enlace e todos os nós atingíveis utilizando esse enlace precisam ser alterados. Desta forma, a entrada do nó B é atualizada conforme novos dados recebidos (a entrada do nó A não possui alteração). Os nós C e E realizam processamento similar, resultando nas tabelas ilustradas em (B).

Após essa atualização, os nós C, D e E enviam para seus vizinhos as tabelas atualizadas, resultando numa alteração das tabelas de A, B, D e E, ilustrada em (C). Novamente esses nós enviam suas tabelas para seus vizinhos, resultando em outra alteração das tabelas de A, B e C, ilustrada em (D). Esses nós novamente enviam suas tabelas, porém não há mais alteração, e o algoritmo novamente convergiu. As tabelas de rotas estão novamente atualizadas para o roteamento.

Em todos os casos até agora, é assumido que o custo de cada aresta é igual a um. O algoritmo, porém, funciona para arestas com custo associado, ou peso, diferente de um. Nesse caso, em lugar de aumentar o custo da tabela enviada em apenas uma unidade, aumenta-se o custo à proporção do custo avaliado para o enlace.

Como foi verificado, o algoritmo de Bellman-Ford possui um funcionamento bastante simples, mas está sujeito a algumas inconveniências. Entre elas encontra-se o chamado efeito *bouncing*, no qual o algoritmo executa iterativamente atualizações de aumento no

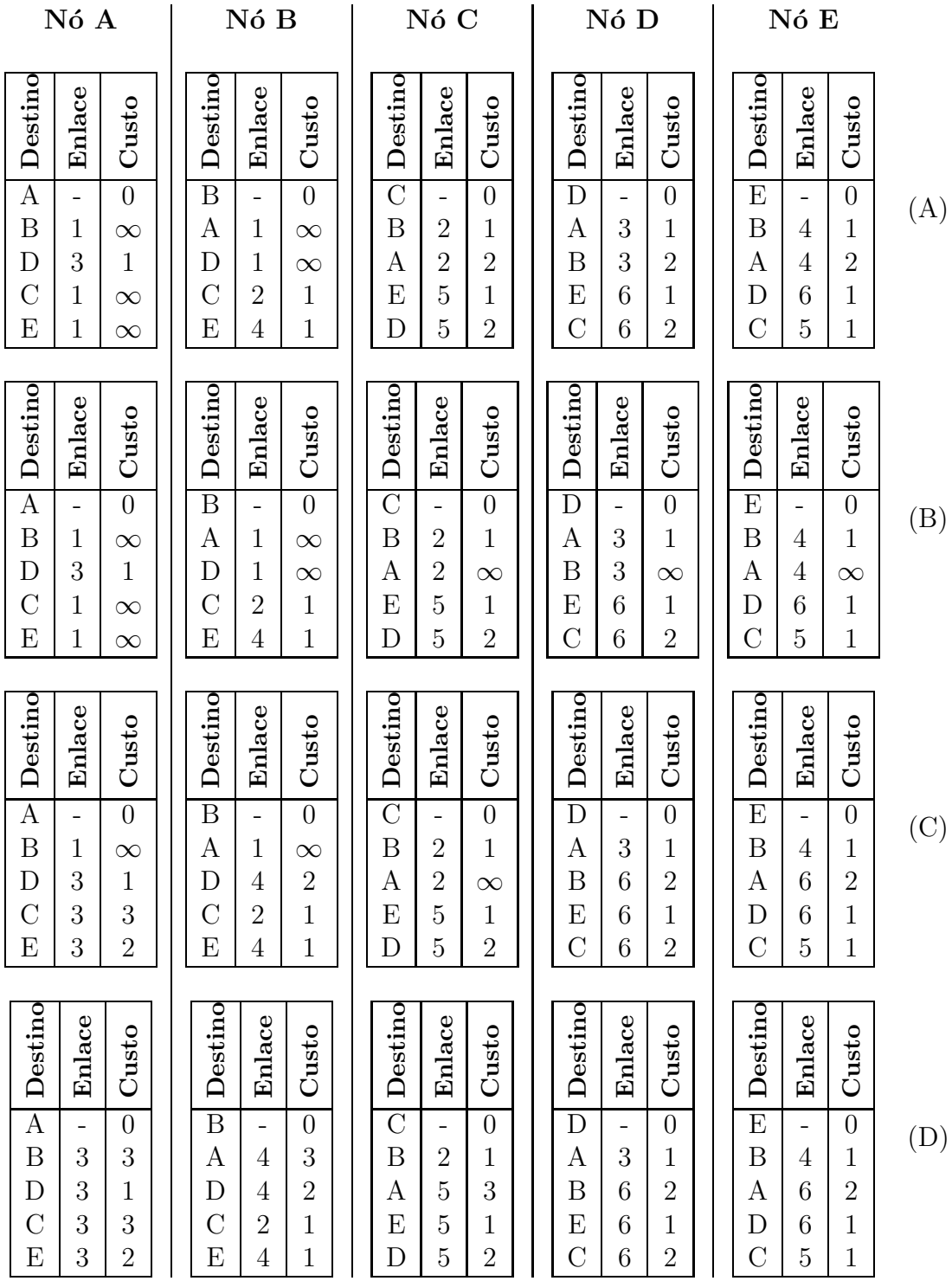

Figura 2.4: Tabelas de vetor-distância do algoritmo de Bellman-Ford após falha do enlace 1.

custo das tabelas até atingir o valor desejado, levando mais tempo para convergir que o desejado.

Outra inconveniência do algoritmo é o efeito denominado contagem até infinito, que ocorre quando um par de nós fica incomunicável (em todos os caminhos que ligam um nó ao outro, arestas são removidas ou ficam falhas). Esse efeito, em algumas situações, acarreta numa atualização iterativa de aumento do custo na tabela de rotas de determinados nós, tendendo a infinito. Como essa convergência não ocorre em tempo finito, as mensagens de envio de tabelas de rotas ficam por tempo indeterminado sendo enviadas, causando congestionamento na rede e impossibilitando comunicações efetivas na rede.

Há outros eventos que podem ocorrer que acarretam em erros na tabela de roteamento. Tendo em vista que o tempo entre o recebimento de uma atualização e o envio da tabela para seus vizinhos, na prática, não é nulo, os nós podem receber outras informações que podem levar a uma incoerência nas tabelas, conforme descrito em [16].

Diversas alterações foram feitas no algoritmo original tendo em vista contornar as inconveniências mencionadas. Essas alterações estão implementadas nos diversos protocolos que utilizam o algoritmo, como o RIP versões 1 e 2 e o RIPng, este utilizado em redes com IPv6 (protocolo IP em sua versão 6).

2.1.1.2 Roteamento pelo Caminho mais Curto (Dijkstra)

Uma outra tecnologia de roteamento, denominada estado de enlace, foi desenvolvida para uso na plataforma Arpanet [17, 18]. Em lugar de trocar informações de distâncias até os destinos, os nós mantém um mapa, contendo a topologia completa da rede, atualizada somente quando há uma alteração na topologia. Esse mapa, normalmente representado por um grafo, pode ser utilizado para identificar rotas mais precisas que as encontradas com os protocolos baseados em vetor-distância. De fato, as rotas podem ser tão precisas como se as mesmas fossem calculadas de forma centralizada, porém a computação é feita de forma distribuída.

Diversos protocolos se utilizaram da nova tecnologia de roteamento introduzida para a Arpanet. O protocolo IS-IS foi desenvolvido para a camada de rede seguindo a estrutura

OSI. O protocolo OSPF foi desenvolvido pela IETF para uso na Internet, e é recomendado pela IAB como um substituto do protocolo RIP.

Os protocolos de roteamento pelo caminho mais curto, como o OSPF e o BGP, utilizam a tecnologia de estado de enlace para obter os melhores caminhos até os destinos. Esses protocolos se baseiam no algoritmo proposto por E. W. Dijkstra [19], conhecido como *shortest path first* (caminho mais curto primeiro, literalmente). O algoritmo de Dijkstra divide os nós (pontos de rede) em dois conjuntos: o conjunto de nós avaliados, que chamaremos de E, para os quais o caminho mais curto já é conhecido, e o conjunto de nós restantes, que chamaremos de R. O algoritmo também mantém uma lista ordenada de caminhos, que chamaremos de O. O algoritmo possui os passos descritos em seguida. [16]

1. Inicialize o conjunto E com apenas a origem s e o conjunto R contendo todos os demais nós. Inicializalize a lista de caminhos O com todas as arestas (enlaces) adjacentes a s. Cada entrada de O deve conter como métrica o custo associado ao enlace correspondente. Ordene O pelo custo.

2. Se a lista O estiver vazia ou se a primeira entrada contiver um custo com valor infinito, marque todos os nós em R como inalcançáveis e termine a execução do algoritmo.

3. Considere p o primeiro caminho encontrado em O. Remova p da lista O. Seja v o nó final de p. Se v já estiver na lista E, então volte ao passo 2. Senão, p é o caminho mais curto até v. Remova v de R e inclua-o em E.

4. Construa um novo conjunto de caminhos possíveis concatenando p a cada um dos enlaces adjacentes a v. São selecionados apenas enlaces que levem a nós em R. O custo associado a cada caminho é a soma do custo de p com o custo associado ao enlace concatenado. Inclua os novos caminhos a O, mantendo a ordenação pelo custo. Volte ao passo 2.

Pode-se facilmente verificar que o número de iterações do algoritmo é equivalente ao número de enlaces do grafo. Desta forma, se a inclusão de arestas nos conjuntos utilizados

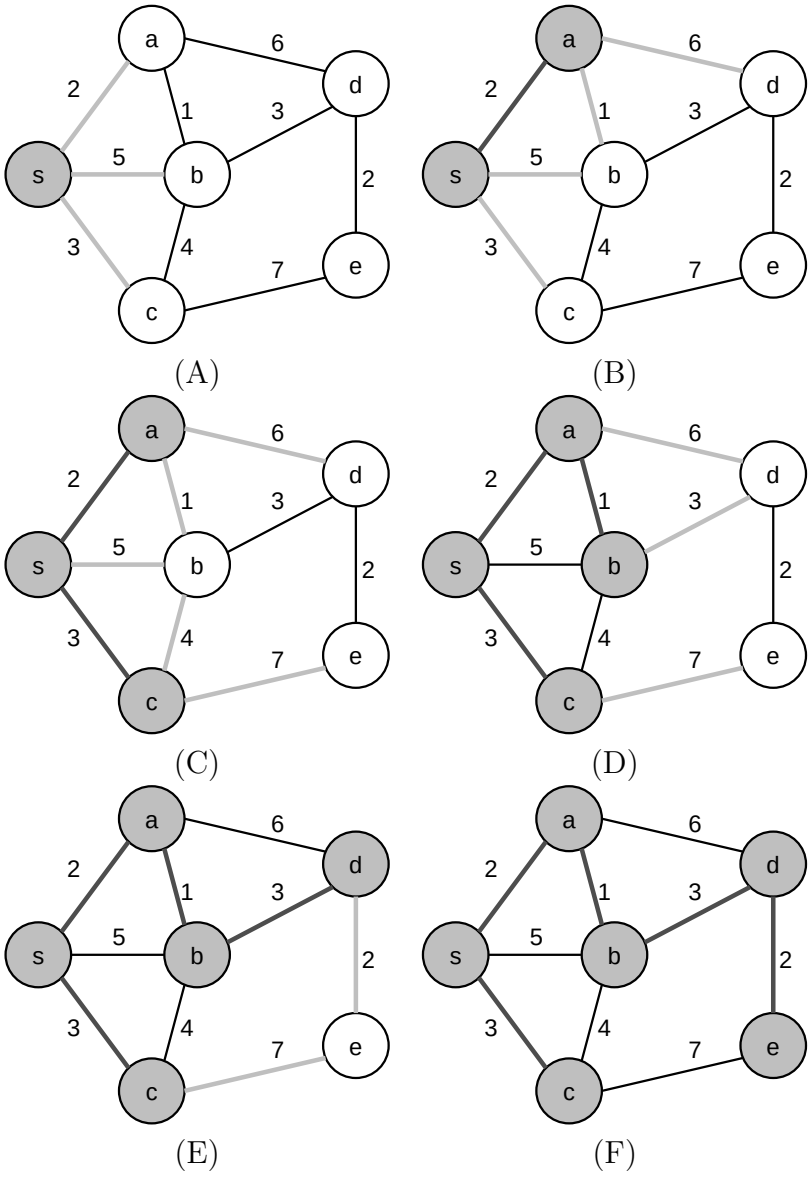

Figura 2.5: Exemplo da execução do algoritmo de Dijkstra.

no algoritmo for feita de maneira adequada, pode-se assumir que esse algoritmo tem complexidade $O(M.\lg M)$, onde M é o número de enlaces. Considerando que o algoritmo de Bellman-Ford converge em $O(M.N)$, com N sendo o número de nós do grafo, verifica-se que o algoritmo de Dijkstra é mais eficiente, possibilidando uma convergência mais rápida do algoritmo.

Um exemplo da execução do algoritmo é demonstrado na figura 2.5. Nessa figura, o grafo inicial, no qual as distâncias de s aos demais nós serão calculadas, é mostrado em (A). Inicialmente, são selecionadas as arestas adjacentes a s e incluídas na lista O. Ordenando-se essa lista, verifica-se que o primeiro caminho a ser selecionado é o caminho

$s - a$, com custo 2. Desta forma, assume-se que o caminho mais curto entre s e a é a aresta $s - a$. Em seguida são selecionadas as arestas adjacentes a a, incluindo em O os caminhos $s - a - b$, com custo 3, e $s - a - d$, com custo 8. O resultado é mostrado em (B).

Continuando a execução, verifica-se que o próximo caminho a ser selecionado pode ser ou o caminho $s - c$, ou o caminho $s - a - b$, ambos com custo 3. Supondo que o caminho selecionado seja $s - c$, identifica-se que o caminho mais curto até c é esse, com custo 3. Novamente são incluídos em O os caminhos das arestas adjacentes: $s - c - b$, com custo 7, e $s - c - e$, com custo 10. O resultado é mostrado em (C).

Na próxima iteração do algoritmo, é selecionado o caminho $s - a - b$, com custo 3. Pode-se observar que esse caminho tem custo inferior à aresta que liga diretamente s a b, que tem custo 5. O caminho $s - a - b$ é, portanto, considerado o caminho mais curto até b. O caminho $s - a - b - d$, com custo 6, é incluído na lista O. Na próxima iteração, o caminho $s - b$, com custo 5, é selecionado, porém é descartado porque b já foi visitado. O resultado nesse passo da execução é mostrado em (D).

O próximo caminho escolhido é o caminho $s - a - b - d$, com custo 6. Esse caminho é considerado o mais curto até d. Nas próximas iterações, os caminhos $s - c - b$, com custo 7, e $s - a - d$, com custo 8, são descartados, visto que seus destinos já foram visitados. O resultado é mostrado em (E). Finalmente, o caminho $s - a - b - d - e$, com custo 8, é selecionado como caminho mais curto até e, descartando em seguida o caminho $s - c - e$, com custo 10. O resultado final é mostrado em (F).

O uso da tecnologia de estado de enlace possui uma série de vantagens em relação ao algoritmo de vetor-distância. Entre as vantagens, pode-se citar uma convergência mais rápida e sem ciclos e o suporte a métricas mais precisas, incluindo, se necessário, o uso de mais de uma métrica. Entretanto, é necessário que todos os nós mantenham a topologia completa da rede [16].

O algoritmo de Dijkstra é executado sem problemas em roteamento interno, no qual a topologia completa da rede pode ser conhecida por cada um dos nós. Porém, para o roteamento na Internet como um todo, os nós não têm o conhecimento completo da

topologia da rede, impossibilitando a execução pura e simples do algoritmo. Uma rede, conforme já mencionado, é geralmente conectada a outras redes do mundo através de um ou mais *gateways* externos, que interligam um sistema autônomo a outros sistemas autônomos da Internet.

Em sistemas autônomos que possuem apenas um único *gateway*, o problema ocasionado pelo desconhecimento da topologia da rede é facilmente resolvido. É necessário apenas definir um caminho padrão até esse *gateway*, o que pode ser feito utilizando tanto algoritmos de vetor-distância quanto de estado de enlace. Porém, quando for necessário escolher entre mais de um *gateway*, a solução do caminho padrão se torna inviável. Nesse caso, são armazenados na tabela de roteamento o que o protocolo OSPF chama de registros de estado de enlace de *gateways*. Isto leva a uma performance melhor para o algoritmo de estado de enlace, visto que as métricas são mais precisas e a computação é mais simples [16].

2.1.2 Protocolos de Roteamento da Internet

Como já foi descrito anteriormente, os protocolos implementam os algoritmos de roteamento em redes reais. Nesta seção são descritos alguns dos protocolos de roteamento mais utilizados na Internet, como o RIP e o OSPF (protocolos de roteamento interno) e o BGP (protocolo de roteamento externo). Também são comentados outros protocolos encontrados na Internet.

2.1.2.1 RIP - *Routing Information Protocol*

O protocolo RIP foi inicialmente desenvolvido como um componente do código de controle de rede para a versão BSD do Unix, incorporado em um programa denominado *route management daemon*. Ele é, em sua primeira versão, um protocolo extremamente simples, e que exige configurações mínimas, que podem ser entendidas e programadas com um mínimo de esforço [16]. O protocolo foi utilizado em diversos produtos e documentado na RFC-1058 por Charles Hedrick [15]. Essa RFC também sugeriu diversos aperfeiçoamentos à implementação inicial.

O protocolo RIP é um protocolo baseado no algoritmo de vetor-distância de Bellman-Ford. Ele difere de outros protocolos baseados neste algoritmo em diversos aspectos como o tipo de métrica, a estrutura de endereçamento e o intervalo de enlaces que suporta. RIP foi desenvolvido como um protocolo de roteamento interno (IGP - *Internal Gateway Protocol*), ou seja, para comunicação entre nós de um mesmo AS [16].

Os endereços presentes nas tabelas RIP são endereços Internet de 32 bits. Uma entrada nas tabelas de roteamento pode representar um ponto de rede, toda uma rede ou uma subrede. Não há uma especificação de tipo de entrada, de forma que, pelo formato do endereço e pela máscara de endereçamento utilizada na subrede específica, identifica-se o que o endereço representa.

Por padrão, o protocolo RIP utiliza uma métrica bastante simples: a distância é o número de enlaces que devem ser utilizados para alcançar o destino, ou seja, a contagem de *hops*. Essa distância é representada como um número inteiro entre 1 e 15, com 16 representando o infinito. A RFC sugere a utilização de outras métricas, entretanto as métricas seriam difíceis de implementar em função do baixo valor da representação de infinito. Um valor de infinito maior, porém, acarretaria em uma latência maior em caso de ocorrência de uma contagem até infinito, descrita na seção 2.1.1.1.

A atualização das tabelas de roteamento é feita através de mensagens que contém a tabela de roteamento do nó de origem para seus vizinhos (*broadcast*). Os pacotes normalmente são enviados a cada 30 segundos, ou mais freqüente em alguns casos específicos. Se uma rota não é atualizada após 180 segundos, ela é considerada falha, e o procedimento de atualização da tabela de rotas é executado.

O processo RIP, ao receber uma mensagem, atualiza sua tabela de roteamento. Cada entrada da tabela de roteamento contém, pelo menos, o endereço do destino, a métrica (custo) associada ao destino, o endereço do "próximo roteador", um *flag* indicando atualização recente e alguns temporizadores. Ao receber uma mensagem, o roteador examina cada uma das entradas enviadas. Diversas validações são feitas nas entradas, e entradas incorretas são ignoradas. Se a entrada possui métrica diferente de infinito, o valor da métrica é acrescido de uma unidade, correspondente ao último *hop*. Em seguida, a

entrada é pesquisada na tabela local de roteamento, e o procedimento do algoritmo de vetor-distância é executado:

- Se a entrada não está presente e a métrica recebida é diferente de infinito, a entrada é incluída, a métrica é inicializada com o valor recebido, o "próximo roteador" é marcado com a origem da mensagem RIP e um *timer* é inicializado para a entrada incluída.

- Se a entrada está presente com uma métrica maior, a métrica e o "próximo roteador" são atualizados e o *timer* para a entrada é reinicializado.

- Se a entrada está presente e o "próximo roteador" corresponde à origem da mensagem RIP, a métrica é atualizada, caso ela seja diferente do valor armazenado e, em todos os casos, o *timer* para a entrada é reinicializado.

- Em qualquer outro caso, a entrada recebida é ignorada.

O protocolo RIP também pode receber requisições de atualização. Uma requisição é normalmente enviada quando um roteador está iniciando, de forma a obter de seus vizinhos um valor inicial para sua tabela de roteamento.

Com o aparecimento de protocolos mais modernos, como o OSPF e o IS-IS, diversos pesquisadores sentiram a necessidade de criar uma segunda versão para o RIP. Essa segunda versão, chamada de RIP-2 foi definida por Gary Malkin em [20]. Esse trabalho se baseou no fato de que o protocolo RIP, apesar de possuir desempenho inferior aos novos protocolos, possuía diversas vantagens, como o baixo *overhead* de largura de banda em redes pequenas e o tempo reduzido de configuração e manutenção.

O protocolo RIP-2 introduz uma série de aperfeiçoamentos no protocolo original, como o roteamento em subredes e o suporte a CIDR, autenticação e transmissão *multicast*. Uma análise do protocolo é apresentada em [21] e a definição da RIP-2 MIB é descrita em [22].

A principal diferença entre RIP-2 e RIP-1 é o formato da mensagem enviada. O protocolo RIP-2 utiliza uma série de campos que não eram utilizados e estavam abertos para uso posterior no protocolo RIP-1. Desta forma, a mensagem enviada no protocolo

RIP-2 contém, além do endereço IP do destino e da métrica, a máscara de subrede e o "próximo roteador", além de campos para integração com protocolos de roteamento externo.

Outra adaptação do protocolo RIP é o protocolo RIPng, definido para uso com o protocolo IPv6. Essa adaptação, basicamente, abrange a alteração do tamanho do endereço para 16 bytes, tamanho utilizado pelo protocolo IPv6, e a utilização da autenticação provida pelo próprio módulo de segurança do IPv6 [23].

2.1.2.2 OSPF - *Open Shortest Path First*

O protocolo OSPF [24] é uma protocolo de estado de enlace, baseado no algoritmo de Dijkstra. O OSPF foi desenvolvido com suporte especial para a distinção entre *hosts* e roteadores, para redes *broadcast*, como a Ethernet [11] e a FDDI [25], e não-*broadcast*, como a X.25 [26] e a ATM [11], e para a separação de redes muito grandes em áreas.

A distinção entre *hosts* e roteadores é utilizada pelo protocolo OSPF para reduzir o número de entradas na tabela de rotas. Tendo em vista que, na maioria das redes, há apenas um roteador que liga todos os nós de uma sub-rede aos demais nós da Internet, é suficiente para os nós externos a essa sub-rede saber o caminho até o roteador, visto que todos os demais nós são alcançáveis através do mesmo. Desta forma, a entrada correspondente a todos os nós da sub-rede é associada ao endereço da sub-rede, indicando o caminho até o roteador da sub-rede.

O suporte do protocolo OSPF às redes *broadcast*, como a rede Ethernet [11], se baseia em duas características dessas redes: a conectividade total, pela qual cada nó pode enviar pacotes para cada um dos outros nós, e a capacidade de *broadcast*, pelo qual um nó pode enviar um pacote que alcance todos os demais nós da rede (*broadcasting*) ou de um grupo (*multicasting*). Nessas redes, considerando n o número de nós da rede, pode-se estabelecer $n(n-1)/2$ adjacências. Cada nó identificará $(n-1)$ enlaces a serem repassados a seus vizinhos, gerando um número grande de mensagens. O protocolo OSPF busca reduzir esse número de mensagens, elegendo na rede um nó responsável por centralizar as comunicações com outros nós da rede. Desta forma, todos os nós recebem e repassam informações

através do nó designado. O nó é escolhido utilizando o protocolo *Hello* (citado adiante), que também é utilizado para identificar seus pares nos enlaces em que está conectado. Um nó designado de *backup* também é escolhido para acelerar a recuperação de uma falha no nó designado principal.

Algumas redes que executam o protocolo IP usam circuitos virtuais para ligar seus roteadores. Na década de 1980, a implementação do protocolo IP sobre redes X.25 [26] foi bastante popular. Esse tipo de rede caiu em certo desuso, porém novas tecnologias como a ATM [11] surgiram com serviços similares e performance superior. Atualmente o uso da tecnologia ATM também está em declínio. As redes ATM diferem das redes Ethernet porque, apesar de oferecer conectividade total entre os nós, não há suporte a um serviço de *broadcasting* ou *multicasting*. O protocolo OSPF aplica o mesmo gerenciamento para as redes ATM. Os roteadores definem um roteador designado e um roteador de *backup*, e as informações de roteamento são trocadas apenas com esses nós. A grande diferença é a ausência de recursos *multicast*, e as informações terão de ser passadas ponto a ponto.

O tamanho da base de dados de estado de enlace, bem como o tempo de computação e o volume de mensagens do protocolo, aumenta significativamente à medida em que a rede cresce. A memória necessária é muito grande, a computação é muito lenta e o *overhead* de transmissão se torna impraticável. Para auxiliar a resolução desse problemas, o protocolo OSPF divide a rede em uma séria de áreas independentes conectadas por um "*backbone* virtual". Em cada uma dessas áreas, os nós incluem em sua tabela de roteamento apenas os estados dos enlaces da área específica, de forma que o custo do protocolo passa a ser proporcional apenas ao tamanho da área escolhida para o nó [16].

O protocolo OSPF faz uso de outros protocolos, utilizados para fins específicos. O protocolo *Hello* é utilizado para a verificação da funcionalidade dos enlaces e para escolha do nó designado em redes com conectividade total. O protocolo de troca (*Exchange*) realiza a sincronização inicial dos dados entre os nós de uma rede e o roteador designado. O protocolo de inundação (*Flooding*) atualiza os dados dos roteadores de acordo com atualização da situação dos enlaces. Um mecanismo auxiliar (denominado *aging*) é utilizado para remover registros desatualizados das tabelas de roteamento. Cada um

destes protocolos, bem como o OSPF, é discriminado em [24].

2.1.2.3 BGP - *Border Gateway Protocol*

Os protocolos descritos nas seções anteriores possuem bom desempenho e custo razoável para o roteamento em um único sistema autônomo. Porém, na Internet, muitas vezes é necessário estabelecer conexões e trocar pacotes entre sistemas autônomos distintos. Para suprir essa necessidade, foram desenvolvidos os protocolos específicos para roteamento externo, ou seja, roteamento entre sistemas autônomos distintos.

Um dos primeiros protocolos desenvolvidos para permitir a troca de informações entre AS's distintos foi o protocolo EGP (*Exterior Gateways' Protocol*) [27]. Esse protocolo foi desenvolvido no início da década de 1980, com o início da organização da Internet em sistemas autônomos. Porém, à medida em que a Internet foi avançando em tamanho e complexidade, as limitações do protocolo EGP se tornaram cada vez mais aparentes. Desta forma, foi desenvolvido um novo protocolo, denominado BGP (*Border Gateway Protocol*) [2]. É possível encontrar roteadores que ainda executam o protocolo EGP, mas esse protocolo já foi quase integralmente substituído pelo protocolo BGP, o protocolo de roteamento externo mais comum na Internet atualmente.

O protocolo BGP foi desenvolvido pelo *Border Gateway Protocol Working Group*, da IETF. Uma primeira versão foi publicada em junho de 1989, na RFC-1105 [28]; a segunda versão foi publicada em junho de 1990, na RFC-1163 [29]; a terceira versão foi publicada em outubro de 1991, na RFC-1267 [30]; a quarta versão foi publicada inicialmente em julho de 1994, na RFC-1654 [31] e revisada em março de 1995, na RFC-1771 [2]. Essas versões são freqüentemente conhecidas por BGP-1, BGP-2, BGP-3 e BGP-4. A última versão é descrita nesta seção, visto se tratar da versão utilizada como padrão na Internet desde 1995.

Um elemento chave do protocolo BGP-4 é o suporte à arquitetura CIDR (*Classless Inter-Domain Routing*) [32], introduzida em 1992 para mitigar o rápido consumo de endereços de rede e combater a explosão das tabelas de roteamento. A estrutura de endereçamento IP, inicialmente, separava o endereço em 8 bits para a rede e 24 bits para

os pontos de rede. À medida em que essa estrutura foi se mostrando insuficiente, foram criadas classes de endereçamento, separando-se os endereços em classes de 8, 16 e 24 bits para a rede (respectivamente classes A, B e C) e o restante para os pontos de rede. Com o rápido crescimento da Internet e a exaustão especialmente dos endereços da classe B, a arquitetura CIDR foi desenvolvida, permitindo uma flexibilidade maior no número de bits utilizados para a rede e para os pontos de rede, possibilitando um uso mais sensato dos endereços de rede.

O protocolo EGP se baseava no conceito de vetor-distância, pelo qual o nó de origem, que escolherá o melhor caminho, não possui conhecimento do caminho que será adotado, mas apenas do enlace a ser utilizado. Diversos caminhos na Internet, porém, podem não ser recomendáveis, como redes comerciais com custo elevado ou redes acadêmicas instáveis. Pelo algoritmo de vetor-distância, não é possível tomar decisões a respeito do caminho a ser utilizado descartando redes não recomendáveis ou com baixa prioridade.

O protocolo BGP utiliza uma estrutura mais completa para possibilitar a escolha de caminhos evitando redes não recomendáveis. Essa estrutura é conhecida por vetor-caminho. Nessa estrutura, cada atualização de informações de roteamento carrega uma lista completa das redes ou AS's atravessados pelo caminho entre a origem e o destino. O BGP também evita a seleção de caminhos em *loop*, visto que é implementado um controle para que um mesmo caminho não atravesse uma mesma rede ou AS mais de uma vez. Além disso, o BGP permite que cada nó utilize as métricas que considerar mais adequadas para escolha dos caminhos.

A estrutura de vetor-caminho, porém, possui uma desvantagem clara: ela aumenta o tamanho das mensagens de roteamento e a memória necessária para execução do protocolo. Essa memória aumenta também com o aumento do número de redes encontradas na Internet, visto que é necessário armazenar as informações de caminho para cada rede.

Para reduzir o impacto do tamanho das mensagens e da memória para execução do algoritmo, a versão 4 do protocolo BGP introduziu o conceito de agregação de rotas. Esse conceito aproveita a estrutura do CIDR para agregar informações similares em mensagens mais concisas. Por exemplo, supondo que um determinado roteador possui a informação

de que, para alcançar a rede 197.8.0/23 (ou seja, os nós em que os primeiros 23 bits correspondem a 197.8.0) é necessário passar pelo caminho 1 que inicia em "T"; para alcançar a rede 197.8.2/24 é necessário passar pelo caminho 2 que inicia em "T, X"; e para alcançar a rede 197.8.3/24 é necessário passar pelo caminho 3 que inicia em "T, Y". Este roteador tem a possibilidade de agregar essas informações, repassando apenas a mensagem que a rede 197.8.0/22 (que corresponde a todas as redes indicadas acima) é alcançada pelo caminho 4 que inicia em "$T, (X ou Y)$". Esses caminhos podem ser agregados recursivamente à medida em que o algoritmo é executado.

O protocolo BGP possui diversas outras funcionalidades desenvolvidas para suprir as necessidades da comunicação mundial. A Internet, porém, tem crescido em complexidade e tamanho, exigindo novas tecnologias que permitam o uso mais inteligente dos recursos disponíveis e aumentem a eficiência da comunicação na Internet. Diversos trabalhos têm sido propostos para possibilitar o avanço destes processos. Este assunto, porém, continua em estudo.

2.1.2.4 Outros Protocolos de Roteamento

Diversos outros protocolos de roteamento são conhecidos, embora não sejam tão comumente utilizados quanto o RIP, o OSPF e o BGP. Esses outros protocolos não possuem uma utilização tão ampla quanto os protocolos descritos acima, ou se restringem a áreas específicas de conhecimento.

Com o surgimento da plataforma OSI (*Open System Interconnection*) [11], foi desenvolvido um protocolo de roteamento integrado a essa plataforma denominado IS-IS [33]. Esse protocolo é bastante similar ao protocolo OSPF. De fato, ambos os protocolos foram desenvolvidos em paralelo, por equipes diferentes: o protocolo OSPF foi desenvolvido por equipes da IETF baseados na arquitetura da Internet, enquanto que o protocolo IS-IS foi desenvolvido pela equipe que criou a plataforma OSI. Basicamente o protocolo IS-IS é baseado na estrutura de rede fundamentada por esta plataforma, desenvolvida pela ISO para padronização da estrutura da Internet, porém sem grande sucesso.

Outra organização que propôs um protocolo de roteamento na Internet foi a Cisco.

Essa organização propôs um protocolo proprietário denominado IGRP [34], criado como um substituto ao protocolo RIP. No momento da concepção desse protocolo, o protocolo OSPF ainda não estava disponível, e o protocolo RIP, opção mais utilizada na época, possuía uma série de defeitos conhecidos. O protocolo IGRP se baseia na estrutura de vetor-distância do protocolo RIP, porém corrigindo diversos dos defeitos do mesmo e adicionando alguns recursos, como a proteção contra *loops*, o suporte ao roteamento *multipath* e o suporte a rotas padrões. Alguns desses recursos foram patenteados pela Cisco.

Alguns anos após o desenvolvimento do protocolo IGRP, com o surgimento do OSPF, a Cisco desenvolveu um aprimoramento de seu protocolo, futuramente denominado EIGRP (*Enhanced* IGRP) [35]. Esse último protocolo traz uma série de ajustes e melhorias em relação ao protocolo IGRP.

Existem outros protocolos disponíveis na Internet, que não serão descritos nesse trabalho pelo fato de não terem uma abrangência ou uma utilização significativa. Alguns desses protocolos são o roteamento por inundação (*flooding*), o roteamento hierárquico, o roteamento por difusão e o roteamento *multicast* [16].

2.2 Trabalhos Relacionados

Esta seção descreve alguns casos que têm despertado o interesse de pesquisadores em relação a novas políticas de roteamento. A seção 2.2.1 descreve algumas políticas para roteamento em redes sem fio *ad-hoc*. A seção 2.2.2 descreve algumas estratégias descritas para o roteamento baseado em QoS. A seção 2.2.3 descreve algoritmos utilizados para roteamento tolerante a falhas.

2.2.1 Roteamento em Redes sem Fio *Ad Hoc*

Redes sem fio *ad hoc* possuem características que exigem tratamentos específicos para o roteamento. Nesse tipo de rede, os nós não possuem um conhecimento *a priori* da topologia da rede, e, portanto, esses nós precisam descobri-la. A idéia básica é a de que

um novo nó anuncia sua presença e ouve anúncios em *broadcast* de seus vizinhos. Um nó toma conhecimento de novos nós próximos e de caminhos para alcançá-los, e anuncia aos demais que também é capaz de alcançar esses novos nós. Após algum tempo, cada nó conhece todos os demais nós e um ou mais caminhos para alcançá-los [36].

Tendo em vista as limitações das redes sem fio e dos equipamentos que utilizam essas redes, os algoritmos de roteamento em redes *ad-hoc* devem [36]:

- manter a tabela de roteamento suficientemente pequena;
- escolher a melhor rota para um destino dado, que pode ser a mais rápida, a mais confiável, a que possui maior taxa de transferência (*throughput*) ou a mais barata;
- manter a tabela atualizada quando nós "morrem", se movem ou se aproximam;
- exigir o mínimo possível de mensagens, processamento e tempo de convergência.

Diversos algoritmos têm sido propostos para o roteamento em redes *ad hoc*. Para facilitar a organização dos algoritmos, os mesmos foram agrupados conforme características similares. Os algoritmos pró-ativos (ou orientados por tabela – *table driven*) mantêm tabelas com rotas programadas para cada um dos destinos da rede [37, 38, 39, 40]. Os algoritmos reativos (ou sob demanda) descobrem novas rotas para cada destino quando essas rotas são necessárias [41, 42, 43, 44]. Os algoritmos híbridos buscam juntar as melhores características dos algoritmos pró-ativos e reativos, buscando maximizar o resultado do roteamento [45].

Alguns algoritmos criados para o roteamento em redes *ad hoc* buscam suprir necessidades específicas desses tipos de rede. Algoritmos hierárquicos buscam organizar os nós da rede, utilizando técnicas como endereçamento dinâmico dos nós, de forma a facilitar o roteamento [46, 47, 48]. Algoritmos geográficos realizam o roteamento de acordo com o posicionamento geográfico dos nós [49, 50, 51]. Algoritmos *power aware* utilizam técnicas para reduzir o consumo de energia na transmissão das mensagens, tendo como desvantagem a introdução de um atraso [52]. Há também outros algoritmos específicos para transmissão em *multicast* [53, 54], roteamento baseado em QoS [55] e outros.

2.2.2 Roteamento Baseado em QoS (*Quality of Service*)

Os protocolos de roteamento atuais na Internet são otimizados para uma única métrica, como a contagem de hops. Caminhos alternativos com custo aceitável mas não ótimo não são utilizados no roteamento de pacotes na Internet [56]. Desta forma, conexões que exigem uma qualidade mínima e relativamente constante do serviço, como aplicações multimídia e aplicações de alto risco, podem não receber a qualidade necessária para um funcionamento apropriado.

O roteamento baseado em QoS busca estender os paradigmas de roteamento atuais em diversas formas [56]. Em primeiro lugar, para dar suporte a pacotes que necessitam de uma qualidade constante de conexão, múltiplos caminhos entre pares de nós devem ser calculados. Algumas classes de serviços requerem ainda a distribuição de métricas de roteamento adicionais, como o atraso e a largura de banda disponível. Se alguma dessas métricas se altera com freqüência, atualizações de rota podem tornar-se mais freqüentes, consumindo largura de banda e tempo de processamento nos roteadores.

Em segundo lugar, as políticas de roteamento atuais alteram o tráfego de um caminho para outro à medida em que um caminho "melhor" é encontrado. O tráfego é alterado mesmo que o caminho existente dê suporte às necessidades do serviço. Se o cálculo de rotas gera um empate em recursos consumíveis que se alteram com freqüência, como a largura de banda disponível, essa alteração de caminho será mais freqüente e pode introduzir oscilações à medida em que o tráfego alterna entre diversos caminhos alternativos. Além disso, alterações freqüentes de rotas podem aumentar a variação no atraso verificado por usuários finais.

A função básica do roteamento baseado em QoS é encontrar um caminho que satisfaça um conjunto de restrições. Além disso, grande parte dos algoritmos de roteamento por QoS busca otimizar a utilização dos recursos. O roteamento QoS não é trivial, visto que múltiplas restrições usualmente tornam o problema de rotear um pacote inviável, e o dinamismo dos estados de uma rede torna difícil manter informações atualizadas sobre estados e métricas em uma rede de grande porte [57].

Os algoritmos de roteamento baseado em QoS buscam garantir a capacidade de so-

brevivência (*survivability*) de uma rede, que é definida como a habilidade de prover um conjunto essencial de serviços mesmo com a ocorrência de falhas ou ataques [58]. Para prover essa capacidade, é necessário reservar recursos em um caminho apropriado durante toda a duração de uma conexão. Um caminho apropriado é um caminho com resursos suficientes para satisfazer os requisitos de QoS de uma conexão, satisfazendo alguns critérios adicionais como a eficiência global na utilização dos recursos da rede [59, 60, 61, 62, 63].

Ainda que os recursos necessários para uma conexão com QoS tenham sido reservados, uma conexão pode experimentar uma redução na QoS ou até mesmo ser rompida em virtude de uma alteração ou falha de um nó ou aresta no caminho. Para garantir a capacidade de sobrevivência em uma conexão na ocorrência desse tipo de evento, é necessário restaurar rapidamente a conexão através de um caminho apropriado alternativo. Essa recuperação é usualmente chamada de restauração de QoS (*QoS restoration*) [64, 58].

As técnicas de roteamento baseado em QoS podem ser divididas em reativas e pró-ativas [58]. Em algoritmos de roteamento pró-ativos, são reservados recursos para o caminho utilizado e também para caminhos alternativos. Assim, em caso de necessidade da restauração de QoS, um caminho adicional com recursos disponíveis já está disponível. O roteamento reativo reserva os recursos necessários apenas no caminho em uso, buscando um novo caminho apropriado quando houver necessidade de restauração.

2.2.3 Roteamento Tolerante a Falhas

Diversas abordagens têm sido propostas para evitar a perda de pacotes e conexões durante a latência de convergência dos protocolos de roteamento. Chandrashekar et.al. [3] propõem um mecanismo para análise das dependências de caminhos, de forma a reduzir a exploração de novos caminhos e, por conseqüência, reduzir a latência de convergência. Pei et.al. [4] propõem uma série de asserções a serem aplicadas nas redes, com o propósito de comparar caminhos similares e eliminar caminhos inviáveis. Wang et. al. [5] citam uma estratégia baseada em um mecanismo escalável, denominado FRTR (*Fast Routing Table Recovery*), para detecção e recuperação de inconsistências nas tabelas de roteamento do protocolo BGP.

Uma abordagem para aumentar a tolerância a falhas dos caminhos escolhidos, proposta em [6, 7], é baseada na identificação de nós altamente conexos na rede, com o intuito de encontrar caminhos que passem por esses nós. Nós conexos são nós que possuem grande quantidade de conexões com outros nós da rede. Propõe-se que, em caso de ocorrência de falhas na conexão, se encontre um desvio que passa pelo nó altamente conexo de forma a chegar ao destino. Há uma ênfase na conectividade individual dos nós altamente conexos, sem que haja uma preocupação com a conectividade dos demais nós do caminho escolhido para a comunicação.

Capítulo 3

O Algoritmo Proposto

Este capítulo apresenta a descrição do algoritmo de roteamento dinâmico tolerante a falhas proposto neste trabalho. Esse algoritmo é tolerante a falhas no sentido de que os caminhos são escolhidos de forma a possibilitar o uso de alternativas em caso de falhas. O algoritmo é dinâmico no sentido de que o caminho é escolhido à medida em que o mesmo é percorrido, sem uma definição inicial do caminho final que será utilizado.

O restante deste capítulo está organizado como segue. A seção 3.1 descreve o funcionamento do algoritmo e suas principais características. A seção 3.2 apresenta as principais definições e métricas utilizadas como critérios para o algoritmo proposto. A seção 3.3 apresenta a especificação formal do algoritmo, enquanto a seção 3.4 mostra exemplos da execução desse algoritmo. A seção 3.5 apresenta a avaliação das arestas, baseada em critérios como fluxo máximo e comprimento. Finalmente, a seção 3.6 apresenta a formalização das provas do funcionamento do algoritmo.

3.1 Descrição do Algoritmo

Esta seção apresenta o funcionamento básico do algoritmo, bem como as características mais relevantes do mesmo.

O algoritmo de roteamento proposto nesse trabalho recebe como entrada um par de nós da rede, correspondentes à origem e ao destino de uma mensagem a ser enviada. O algoritmo, então, é executado em cada nó da rede, iniciando pelo nó de origem, escolhendo

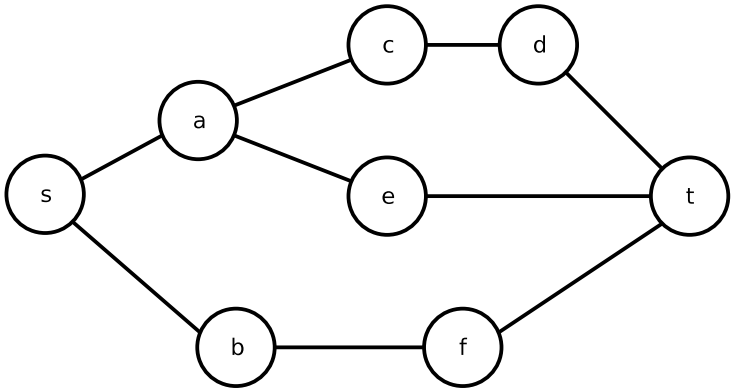

Figura 3.1: Um grafo utilizado como exemplo para o algoritmo.

o próximo nó da rota, dentre os seus nós vizinhos. Quando a mensagem chega a um nó escolhido, esse nó executa o mesmo algoritmo para escolher o nó seguinte, e assim por diante, até que o destino seja alcançado.

A abordagem desse algoritmo é similar à adotada pelo algoritmo de Bellman-Ford, no sentido de que apenas o próximo nó para comunicação é escolhido. Uma das principais vantagens dessa abordagem é o funcionamento do mesmo ainda que a topologia conhecida pela origem não possua as alterações mais recentes da topologia real da rede.

Pelo algoritmo proposto, cada nó da rede escolhe o próximo nó da rota avaliando cada aresta adjacente ao mesmo, avaliação esta baseada em um compromisso (*trade-off*) entre redundância e distância dos caminhos da aresta avaliada. Após avaliar as arestas, a aresta com melhor avaliação é escolhida.

As fórmulas e equações para o cálculo das métricas utilizadas nesse trabalho assumem que cada nó possui uma representação local da topologia da rede. Essa representação da topologia, porém, não é necessariamente completa e atualizada. A representação é feita através de uma estrutura de grafos direcionados, com um conjunto de nós e um conjunto de arestas. Essa estrutura é atualizada periodicamente através de troca de mensagens, conforme descrito na seqüência desse trabalho. Como exemplo da representação da rede em cada nó, suponhamos a rede ilustrada na figura 3.1. Em cada nó haverá uma representação da topologia com os seguintes conjuntos: o conjunto de nós $V = \{s, a, b, c, d, e, f, t\}$ e o conjunto de arestas $E = \{(s, a), (s, b), (a, c), (a, e), (a, s), (b, f), (b, s), (c, a), (c, d), (d, c), (d, t), (e, a), (e, t), (f, b), (f, t), (t, d), (t, e), (t, f)\}$.

A precisão do algoritmo proposto está diretamente relacionada com a correção desta lista. À medida em que a topologia da rede é alterada, todos os nós da rede devem atualizar suas tabelas de acordo com a alteração da topologia. Porém, visto que um dos focos deste trabalho é a tolerância a falhas, o algoritmo funciona mesmo que as tabelas nos nós não estejam atualizadas, ou possuam posições incorretas da topologia da rede.

Os critérios utilizados para avaliação de caminhos para roteamento utilizam como base para o cálculo a própria estrutura de grafos, utilizada para a representação local da topologia da rede. Para o cálculo dos critérios é usada uma cópia da representação da topologia, porém removendo o próprio nó que realiza a avaliação dos critérios, assim como os nós já visitados pela mensagem. Assim, são avaliados os nós que sejam adjacentes ao nó que realiza a avaliação na topologia original, mas que não foram visitados pela mensagem.

Um parâmetro bastante utilizado em algoritmos de roteamento é a distância das rotas. Uma definição simplificada para a distância entre dois nós é o número de arestas da rota com menor comprimento. Grande parte dos algoritmos de roteamento utiliza a distância como a única ou a principal métrica para a seleção da melhor rota para o destino fornecido. Porém, aplicações críticas podem preferir utilizar uma rota mais longa, se a confiabilidade e/ou a robustez forem comprovadamente maiores nesta rota.

Neste trabalho, a distância é utilizada como um critério secundário para a escolha da melhor rota. Entretanto, como mencionado anteriormente, a origem do roteamento não escolhe *a priori* o caminho inteiro da origem ao destino, mas apenas o próximo nó. Desta forma, o uso direto do conceito de comprimento da rota não é aplicável nesse trabalho, visto que a aresta avaliada para o envio da mensagem pode estar vinculada a mais de uma rota. Logo, para a avaliação de uma aresta, é utilizado o menor caminho que passa pela aresta. Por exemplo, na rede representada pela figura 3.1, caso s precise enviar uma mensagem para t, a aresta (s, a) será avaliada. Essa aresta possui duas rotas associadas ao destino t, cada uma com um comprimento diferente. Neste caso, para a avaliação da aresta (s, a) será utilizado o menor caminho que passa por essa aresta.

O principal critério para seleção de caminhos introduzido nesse trabalho é o corte mínimo (ou fluxo máximo) entre o nó avaliado e o destino [8]. Esse critério é utilizado

para avaliar a redundância do caminhos, visto que, quanto maior o fluxo máximo, maior a quantidade de caminhos disjuntos e, por conseqüência, maior é o número de desvios que podem ser utilizados em caso de falha. Por exemplo, na rede representada pela figura 3.1, se o caminho escolhido pelo nó s para chegar ao destino t fosse a partir da aresta (s, a), em caso de falha da aresta (a, e), o nó a possui outras alternativas, como a aresta (a, c), que ainda possui caminhos até o destino.

O algoritmo prevê uma atualização periódica das tabelas de roteamento em cada nó. Essa atualização é feita através de mensagens, cada uma contendo um conjunto de triplas contendo uma aresta, a situação (funcional ou falho) da aresta e um contador para ordenação das situações (*timestamp*). As mensagens contém apenas as alterações ocorridas na topologia e recebidas ou observadas pelo nó que envia a mensagem. Quando um nó recebe uma mensagem de atualização, ele atualiza sua representação local da rede, incluindo ou removendo as arestas indicadas na mensagem. As mensagens são enviadas mesmo que não haja alterações na topologia, para fins de verificação da funcionalidade da aresta. Caso nenhuma mensagem seja recebida após um período de tempo (*timeout*), a aresta é considerada falha, e a informação da falha é marcada para ser enviada a outros nós. Caso uma mensagem seja recebida através de uma aresta considerada inexistente ou falha, a aresta passa a ser considerada funcional, e a informação da nova aresta é marcada para ser enviada a outros nós.

3.2 Definições Preliminares

Esta seção apresenta a formalização de algumas definições utilizadas nas seções seguintes para especificar o algoritmo proposto neste trabalho.

Definição 3.1 (Grafo, Nó e Aresta): Um grafo (ou rede) direcionado G é um par (V, E) de conjuntos, em que V é um conjunto de elementos denominados nós (ou vértices) e E é um conjunto de elementos denominados arestas (ou enlaces), tal que $E \subset \{(v_1, v_2) \in V^2 | v_1 \neq v_2\}$, i.e. cada aresta é um par ordenado de exatamente dois nós diferentes.

O grafo é utilizado neste trabalho para representar a topologia de uma rede. Cada nó

de uma rede é representado por um nó do grafo, e cada enlace da rede é representado por duas arestas no grafo, uma em cada sentido.

Definição 3.2 (Adjacência): Seja $G = (V, E)$ um grafo; sejam $u, v \in V$ nós do grafo G; e seja $e \in E$ uma aresta neste grafo. O nó v é dito adjacente ao nó u se $\exists e' \in E, e' = (u, v)$. A aresta e é dita adjacente ao nó u se $e = (u, u')$.

Definição 3.3 (Caminho): Seja $G = (V, E)$ um grafo; sejam $s, t \in V$ nós do grafo G. Um caminho entre a origem s e o destino t é uma seqüência de nós $p = (v_0, v_1, \ldots v_{n-1}, v_n)$, com $v_0 = s$ e $v_n = t$, tal que:

$$\forall i, 0 \leq i < n, (v_i, v_{i+1}) \in E$$

Dizemos que um nó $v \in V$ pertence a p se $v \in \{v_0, v_1, \ldots v_n\}$. Dizemos que $e \in E$ pertence a p se $\exists i, 0 \leq i < n, e = (v_i, v_{i+1})$.

Em algumas definições e especificações deste trabalho, a palavra *rota* também é utilizada como um sinônimo para caminho. Este termo é normalmente utilizado em tópicos relacionados a redes de computadores, para especificar o caminho utilizado para enviar uma mensagem de um nó de origem para um nó de destino.

Definição 3.4 (Caminhos cíclico e sem ciclos): Seja $G = (V, E)$ um grafo; seja $p = (v_0, v_1, \ldots v_n)$ um caminho neste grafo. O caminho p é dito sem ciclos se:

$$\forall i, j, 0 \leq i < j \leq n, v_i \neq v_j$$

O caminho p é dito cíclico se:

$$\exists i, j, 0 \leq i < j \leq n, v_i = v_j$$

Definição 3.5 (Comprimento de um caminho): Seja $G = (V, E)$ um grafo; seja $p = (v_0, v_1, \ldots v_n)$ um caminho neste grafo. O comprimento do caminho p é o número de elementos da seqüência de nós (n). Para caminhos sem ciclos, o comprimento do caminho

p, ou $|p|$, pode ser definido como o número de arestas deste caminho, ou como o número de nós deste caminho menos um.

Definição 3.6 (Grafo conexo): Seja $G = (V, E)$ um grafo; sejam $u, v \in V$ nós do grafo G. Diz-se que u e v são conexos se $\exists p \in P_{u,v}$, ou seja, existe um caminho entre u e v no grafo G. O grafo G é dito conexo se para todo $u', v' \in V$, u' e v' são conexos.

Definição 3.7 (Fluxo máximo): Seja $G = (V, E)$ um grafo; seja $c : E \to \Re$ uma função correspondente à capacidade das arestas do grafo; sejam $u, v \in V$ nós do grafo G. Um fluxo entre u e v é uma função $f : E \to \Re$ tal que:

$$\forall e \in E, f(e) \leq c(e)$$

$$\forall t \in V - \{u, v\}, \sum_{e=(t,t')\in E} f(e) = \sum_{e=(t',t)\in E} f(e)$$

Dizemos que o tamanho (ou cardinalidade) de um fluxo f, denotado por $|f|$, é igual a:

$$|f| = \sum_{e=(u,t)\in E} f(e) - \sum_{e=(t,u)\in E} f(e)$$

Dizemos que um fluxo f é máximo se para todo fluxo f' entre o mesmo par de nós, $|f| \leq |f'|$.

Definição 3.8 (Corte mínimo): Seja $G = (V, E)$ um grafo; sejam $u, v \in V$ nós do grafo G. Um corte entre u e v é um conjunto de arestas C tais que, removendo todas as arestas em C do grafo G, u e v deixam de ser conexos. Dizemos que o tamanho (ou cardinalidade) de um corte C, denotado por $|C|$, é igual à cardinalidade do conjunto de arestas. Um corte C é dito mínimo se, para todo corte C' entre o mesmo par de nós, $|C| \leq |C'|$.

Diversos artigos provam que, para qualquer par de nós da rede, o fluxo máximo e o corte mínimo possuem a mesma cardinalidade, e podem ser calculados utilizando os mesmos algoritmos [11, 8]. Desta forma, o restante do trabalho utiliza os dois conceitos em conjunto.

Definição 3.9 (Roteamento): Seja $G = (V, E)$ a representação em um grafo de uma

rede; sejam s e t dois nós de G, correspondentes respectivamente à origem e ao destino. Se s precisa enviar uma mensagem para t, é chamado um processo de roteamento, que escolhe um caminho de s a t.

Um processo de roteamento pode ser executado inteiramente em s, ou parcialmente em s e parcialmente nos nós seguintes do caminho escolhido. Nesse último caso, o processo de roteamento é dito dinâmico.

Diversos algoritmos de roteamento são conhecidos, e alguns dos mais utilizados são descritos na seção 2.1.1. Dentre os algoritmos descritos, destacam-se o algoritmo de Bellman-Ford como um exemplo de roteamento dinâmico, e o algoritmo de Dijkstra como exemplo de roteamento não dinâmico.

3.3 Especificação do Algoritmo

Uma característica importante dos algoritmos de roteamento dinâmico é o fato de cada nó escolher apenas parte do caminho utilizado para enviar uma mensagem de uma origem para um destino. Nesse trabalho, o algoritmo de roteamento é responsável por escolher apenas a próxima aresta do caminho até o destino. Cada aresta deve ser escolhida cuidadosamente, de forma a evitar ciclos (*loops*), aumentar a confiabilidade e a robustez e reduzir o comprimento da rota, sempre à medida do possível.

Para escolher uma aresta entre as arestas disponíveis para enviar a mensagem, o roteamento proposto nesse trabalho avalia cada uma das arestas utilizando o cálculo de compromisso (*trade-off*) descrito na seção 3.5. Esse cálculo de compromisso busca avaliar alguns critérios relacionados tanto à redundância quanto ao comprimento dos caminhos correspondentes a cada aresta, dando maior ou menor prioridade conforme uma parametrização do algoritmo.

Suponha que s e t são nós de uma rede, correspondentes respectivamente à origem e ao destino de um processo de roteamento. O processo de envio de mensagem da origem até o destino é descrito na figura 3.2. Nesse algoritmo, é assumido que, quando s quer mandar uma mensagem para t, ele chama o procedimento **SendMessage**($payload$,s,t,$needsAck$), passando o conteúdo da mensagem ($payload$), a origem(s), o destino (t) e a indicação

```
SendMessage(payload, s, t, needsAck)
 1  Let msg.Payload ← payload
 2  Let msg.Src ← s
 3  Let msg.Dest ← t
 4  Let msg.NeedsAck ← needsAck
 5  Let msg.Visited ← an empty set
 6  Run RouteMessage(msg, s)
```

```
ReceiveMessage(msg, u, l)
 1  If msg.Dest = u Then
 4    If msg is an update message Then
 5      Run ReceiveUpdateMessage(msg, u)
 6    Else If msg is ACK to an update message Then
 5      Run ReceiveUpdateAckMessage(msg, u, l)
 6    Else
 7      Send msg.Payload to application level
 2    If msg.NeedsAck Then
 3      Run SendMessage(Ack(msg), u, msg.Src, FALSE)
 8  Else
 9    Run RouteMessage(msg, u)
```

```
RouteMessage(msg, node)
 1  Add node to msg.Visited
 2  If there is a link l that goes directly from node to msg.Dest Then
 3    Send message msg through link l
 4  Else
 5    Let CandLinks ← no link
 6    For each link l that is adjacent to node in node.KnownTopology:
 7      If l has no path to msg.Dest Or l goes to a link in msg.Visited Then
 8        Ignore l
 9      Else
10        Add l to CandLinks
11    Let WorkingGraph ← node.KnownTopology
12    Remove all nodes in msg.Visited from WorkingGraph
13    If CandLinks contains at least one link Then
14      Let l ← the link in CandLinks with largest Γ(WorkingGraph, l)
15      Send message msg through l
16    Else
17      Remove last node from msg.Visited
18      If msg.Visited contains at least one node
19        Let l ← the previous node in msg.Visited
20        Send an update message to l
21        Send message msg back to l
22      Else
23        Return Error: There is no route to msg.Dest
```

Figura 3.2: A especificação do algoritmo utilizado para escolher um caminho até o destino.

```
TopologyMaintainer(s)
 1  Let s.KnownTopology ← a graph containing only s
 2  Let s.TimeStamp ← 1
 3  Let s.LastEdgeInformation ← an empty set
 4  Forever
 5     Wait α seconds
 6     For each link l adjacent to s
 7        If timer for l has expired Then
 8           Let i ←< l, FALSE, s.TimeStamp >
 9           Increment s.TimeStamp
10           Remove l from s.KnownTopology
11           Let s.LastEdgeInformation[l] ← i
12           For each node l' adjacent to s
13              Add i to s.EdgeInformationToSend[l']
14     For each link l adjacent to s
15        Let payload ← s.EdgeInformationToSend[l]
16        Let t ← the node for which l points
17        Run SendMessage(payload, s, t, TRUE)
```

Figura 3.3: A especificação do processo executado em cada nó.

se é necessário o envio de uma confirmação por parte do destino (*needsAck*). Quando um nó u recebe uma mensagem msg através de uma aresta l, ele chama o procedimento **ReceiveMessage**(msg,u,l). O procedimento **RouteMessage** é utilizado como auxiliar para os demais procedimentos. O cálculo da função $\Gamma(G, e)$ será discutido na seção 3.5.

Para possibilitar que as informações de rotas disponíveis reflitam as alterações de topologia à medida em que essas ocorrem, há um processo de atualização executado em cada nó. Esse processo é executado a cada α segundos, com α podendo ser parametrizado (esta parametrização será discutida posteriormente). O processo envia, para todos os seus vizinhos, uma lista contendo todos os caminhos conhecidos para todos os destinos possíveis, como é mostrado na figura 3.3. Cada nó inicia a execução do procedimento **TopologyMaintainer**(s) no momento em que entra na rede. Quando um nó t recebe uma mensagem de atualização, enviada pelo procedimento citado, ele chama o procedimento **ReceiveUpdateMessage**(msg,t), descrito na figura 3.4. Quando um nó t recebe uma confirmação de recebimento de mensagem de atualização, ele chama o procedimento **ReceiveUpdateAckMessage**(msg,t,l), descrito na figura 3.5. Caso nenhuma mensagem

```
ReceiveUpdateMessage(msg, t)
 1  Let s ← msg.Src
 2  Let l ← link from t to s
 3  Update the timer for l in t
 4  If t.KnownTopology does not contain node s
 5     Add node s to t.KnownTopology
 6  If t.KnownTopology does not contain link l
 7     Add link l to t.KnownTopology
 8     Let i ←< l, TRUE, t.TimeStamp >
 9     Increment t.TimeStamp
10     Let t.LastEdgeInformation[l] ← i
11     For each node l' adjacent to t
12        Add i to t.EdgeInformationToSend[l']
13     For each edge information i' available in t.LastEdgeInformation
14        Add i' to t.EdgeInformationToSend[l]
15  For each edge information i in msg.Payload
16     Let i' ← t.LastEdgeInformation[i.Edge]
17     If i' does not exists Or i'.TimeStamp <= i.TimeStamp Then
18        Let t.LastEdgeInformation[i.Edge] ← i
19        For each node l' adjacent to t
20           Add i to t.EdgeInformationToSend[l']
21        If i.Working = TRUE Then
22           Add link i.Edge to t.KnownTopology
23        Else
24           Remove link i.Edge from t.KnownTopology
```

Figura 3.4: A especificação do algoritmo executado no recebimento de uma mensagem de atualização.

```
ReceiveUpdateAckMessage(msg, t, l)
1  For each edge information i in the original message of msg
2     Remove i from t.EdgeInformationToSend[l]
```

Figura 3.5: A especificação do algoritmo executado no recebimento da confirmação de uma mensagem de atualização.

seja recebida de um determinado nó após um *timeout* parametrizado, a aresta correspondente é considerada falha. Este *timeout* é chamado de β ($\beta > \alpha$).

Na avaliação das arestas a serem utilizadas no procedimento **RouteMessage**, descrito na figura 3.2, as arestas que levam a nós já visitados pela mensagem (ou seja, nós em $msg.Visited$) são utilizadas apenas quando realmente são necessárias. Esse tratamento é feito para evitar ciclos no caminho percorrido.

Há, porém, um caso específico em que arestas que geram ciclos são consideradas. Levando em conta que as informações nos nós não estão necessariamente atualizadas, um nó pode ter sido escolhido para o roteamento em virtude de caminhos que não existem mais, ou que estão falhos. No momento em que a mensagem alcança um nó que já possua essa informação mais atualizada, esse nó poderá verificar que as únicas arestas que possuem caminhos até o destino levam a nós já previamente visitados. Neste caso, será necessário retornar a mensagem à aresta a partir da qual a mensagem chegou ao nó. Porém, o nó que receber essa mensagem provavelmente terá informações desatualizadas sobre a rede, visto que selecionou para o roteamento um nó sem opções de rotas. Logo, para que esse nó possua informações mais atuais, a mensagem de roteamento programada para ser enviada para esse nó é antecipada, e é enviada antes de retornar a mensagem para o nó. Desta forma, o novo nó poderá tomar a decisão de um novo caminho baseando-se em informações mais atuais de topologia, evitando assim caminhos sem opções de rotas.

3.4 Exemplos de Execução do Algoritmo

A figura 3.6 ilustra o funcionamento do algoritmo proposto neste trabalho. Nessa figura, vamos supor que o nó s precisa enviar uma mensagem para o nó t. O algoritmo de roteamento é executado em s, avaliando todas as arestas adjacentes, ou seja, as arestas (s, a), (s, b) e (s, c), com o objetivo de escolher qual dessas arestas será utilizada no caminho. Supondo que a aresta escolhida seja (s, b), então a mensagem é enviada de s para b.

Quando o nó b recebe a mensagem enviada de s para t, ele avalia todas as suas arestas adjacentes, (b, d), (b, e) e (b, s). Como a aresta (b, s) vai para um nó que já foi visitado,

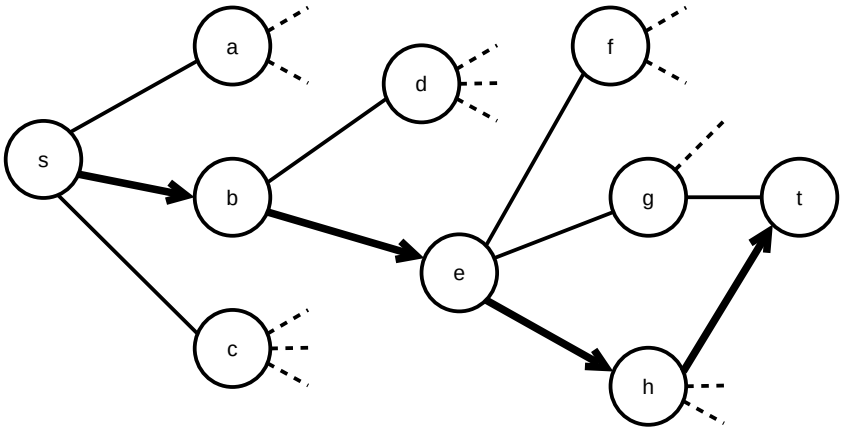

Figura 3.6: Um exemplo de como o algoritmo funciona.

ela é descartada. Suponha que a aresta escolhida seja (b, e). A mensagem é então enviada para o nó e.

O nó e executa o mesmo procedimento de roteamento, avaliando as arestas (e, f), (e, g), (e, h) e (e, b). A aresta (e, b) é descartada, visto que ela vai para b, que já foi visitado. Suponha que a aresta escolhida seja (e, h). A mensagem é então enviada para o nó h.

Finalmente, o nó h avalia suas arestas adjacentes. Como ele possui uma aresta que vai diretamente para o destino final t, as demais arestas são descartadas, e a aresta (h, t) é utilizada. Então, a mensagem é enviada para o nó t, chegando ao destino final.

Considere agora um outro exemplo. Suponha que, na mesma rede descrita acima, ocorre uma falha na aresta (h, t), que é utilizada no caminho entre s e t, conforme descrito. A figura 3.7 ilustra o resultado da rede após essa falha.

Suponha, novamente, que s precisa enviar uma mensagem para t, pouco depois da ocorrência da falha. Suponha que, no momento do envio da mensagem, apenas os nós h e t (vizinhos da aresta falha) têm o conhecimento dessa falha. O algoritmo será executado no nó s, enviando a mensagem para b, que por sua vez envia a mensagem para e, que envia a mensagem para h. O procedimento ocorre da mesma forma que o exemplo anterior nestes nós, visto que suas informações de topologia ainda não foram alteradas.

Quando o nó h receber a mensagem que se destina a t, ele verificará que os únicos caminhos possíveis para t são através de arestas que ligam a nós já visitados, como o nó e. Visto que não há outra alternativa para o envio dessa mensagem, o nó h se prepara para

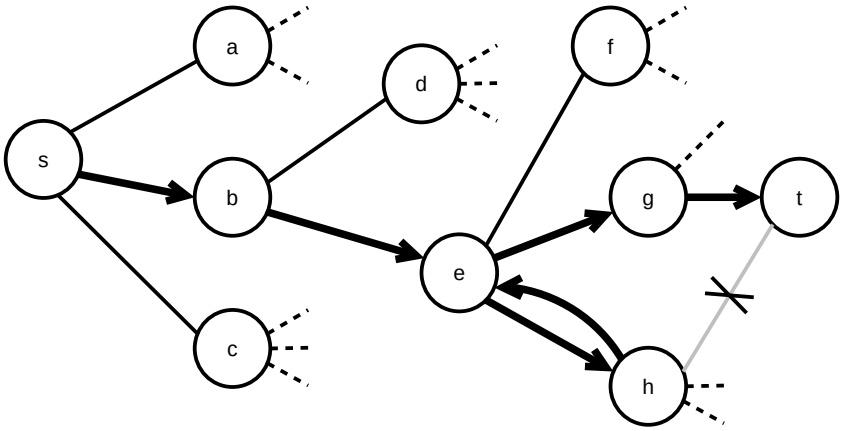

Figura 3.7: Um exemplo de como o algoritmo se comporta em caso de falha.

enviar a mensagem para e, que é o nó imediatamente anterior no caminho. Para evitar a ocorrência de um ciclo, h antecipa o envio da mensagem de atualização de topologia para e, que seria enviada nos próximos α segundos pelo processo **TopologyMaintainer**, descrito na figura 3.3. Com o recebimento dessa mensagem, e estará apto a tomar uma decisão baseado em uma informação mais recente da topologia. Observe que, se essa mensagem não fosse enviada, o nó e continuaria tomando a decisão baseado na mesma tabela de roteamento, e poderia enviar a mensagem para h novamente. Como a mensagem de atualização é enviada, o nó e poderá desconsiderar a aresta falha, e enviar a mensagem para um nó com caminhos disponíveis. Finalmente, após o envio da mensagem de atualização, h envia a mensagem original para e.

Quando o nó e receber novamente a mensagem original, esse tomará uma nova decisão sobre a aresta que deverá ser utilizada. Essa decisão será tomada com base em informações mais recentes dos caminhos disponíveis, tendo conhecimento de que a aresta (h, t) não está funcional. Desta forma, uma outra aresta é escolhida, no caso, a aresta (e, g), levando a mensagem ao nó g. Esse nó envia a mensagem para o nó t através da aresta (g, t), e a mensagem chega ao seu destino.

3.5 Avaliação das Arestas - Redundância *versus* Distância

O roteamento descrito neste trabalho avalia as arestas adjacentes ao nó em que o processo é executado, com o objetivo de escolher a aresta que possui o melhor compromisso entre

redundância e distância, de forma que, através de pesos, pode-se tanto escolher caminhos com maior redundância quanto com menor comprimento.

O cálculo da avaliação das arestas se baseia em uma série de critérios quantitativos relacionados à redundância e ao comprimento dos caminhos relacionados às arestas. Para cada um desses critérios, é associado um peso parametrizável, de forma a possibilitar a ênfase maior ou menor em um critério ou outro.

No grafo utilizado para a avaliação das arestas, são removidos os nós que já foram anteriormente visitados pela mensagem, assim como as arestas adjacentes aos mesmos. Essa remoção evita que caminhos que serão naturalmente desprezados pelos nós seguintes do roteamento sejam utilizados como base para a seleção da aresta avaliada, e assim gerem uma avaliação incorreta da redundância e do comprimento dos caminhos da aresta avaliada.

A equação correspondente ao cálculo da avaliação das arestas está descrita a seguir:

$$\Gamma(G, e) = \sum_{c_n \in C} \omega_n \times c_n(e) \tag{3.1}$$

Nesta equação, $\Gamma(G, e)$ é a função de avaliação (*trade-off*), e é a aresta sendo avaliada, C é o conjunto de critérios (que será descrito na seqüência) e ω_n é o peso associado ao critério c_n.

Os critérios utilizados para avaliação das arestas são funções que, a partir de uma aresta, retornam um valor numérico. Inicialmente são utilizados como critérios: a cardinalidade do fluxo máximo (ou do corte mínimo) entre o nó correspondente à aresta avaliada e o nó de destino (c_1) e o comprimento do menor caminho entre esses nós (c_2). Cada um desses critérios será descrito na seqüência.

Outros critérios podem ser utilizados no futuro para avaliação das arestas para roteamento. Um desses critérios é o número total de caminhos disponíveis entre o nó avaliado e o destino. Outro critério é o comprimento médio desses caminhos. A utilização desses critérios foi avaliada neste trabalho, porém algoritmos triviais para identificação desses valores possuem complexidade não-polinomial, logo não foram utilizados. Trabalhos futu-

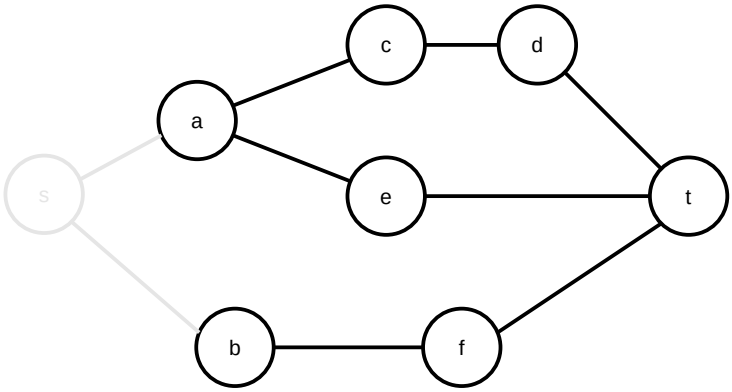

Figura 3.8: Grafo da figura 3.1 com o nó s sombreado.

ros podem também utilizar critérios relacionados a QoS, como o atraso, o custo, a largura de banda, entre outros.

Todos os critérios utilizam a rede ilustrada na figura 3.1 como exemplo. A figura 3.8 ilustra o grafo utilizado para avaliação dos critérios, com o nó s sombreado, que será desprezado pelo algoritmo por ser utilizado como origem.

3.5.1 Critério c_1: Fluxo Máximo

O principal critério utilizado neste trabalho para avaliação das arestas para roteamento é o fluxo máximo (ou corte mínimo) entre o nó correspondente à aresta avaliada e o destino. O fluxo máximo e o corte mínimo são descritos na seção 3.2. Este critério é chamado de c_1.

Conforme descrito anteriormente, a avaliação desse critério é realizada tendo como base o grafo que representa a topologia conhecida pelo nó que realiza a avaliação, removendo os nós já visitados pela mensagem e as arestas adjacentes aos mesmos. Um algoritmo clássico para a avaliação do fluxo máximo é o algoritmo de Ford-Fulkerson [8, 9], descrito a seguir. O algoritmo utiliza a estrutura de grafo valorado, em que, para um grafo $G = (V, E)$ temos uma função $c : E \rightarrow \Re$, que associa cada aresta a um valor. Assume-se que a função $c(e)$ retorna 1 para todas as arestas do grafo.

O algoritmo de fluxo máximo utiliza uma estrutura auxiliar denominada grafo residual. Para um grafo $G = (V, E, c)$, considerando o fluxo f, o grafo residual $G_f = (V, E_f, c_f)$ corresponde a um grafo com os mesmos nós do grafo original, com $E_f = (E \cup E') -$

$\{e|f(e) = c(e)\}$, $E' = \{(v_1, v_2)|(v_2, v_1) \in E\}$ e $c_f\big((v_1, v_2)\big) = c\big((v_1, v_2)\big) - f\big((v_1, v_2)\big) + f\big((v_2, v_1)\big)$. Em termos mais simples, o grafo residual é o grafo original, removendo-se as arestas em que o fluxo seja igual ao custo da aresta e incluindo-se as arestas opostas às arestas do grafo original em que o fluxo for superior a zero.

1. Escolha um fluxo inicial f (normalmente $\forall e, f(e) = 0$);
2. Encontre o grafo residual G_f;
3. Encontre um caminho sem ciclos entre a origem e o destino no grafo residual;
4. Se não for encontrado caminho, retorne f como fluxo máximo;
5. Se encontrar o caminho:
   (a) Seja c_{min} o menor valor de $c(e)$ para todas as arestas do caminho encontrado;
   (b) Para cada aresta (u, v) do caminho encontrado:
      i. $f(u, v) \leftarrow f(u, v) + c_{min}$
      ii. $f(v, u) \leftarrow f(v, u) - c_{min}$
   (c) Repita o algoritmo a partir do passo 2.

A avaliação do fluxo máximo entre um par de nós executa, para capacidades com valores inteiros, no máximo $|f|$ buscas em largura, que possuem complexidade igual a $O(M)$, com M sendo o número de arestas do grafo. Portanto, a complexidade do algoritmo é igual a $O(M|f|)$. Como o fluxo máximo não ultrapassa o grau do nó de origem (considerando que as arestas possuem capacidade 1) e o grau de um nó é, no pior caso, $O(N)$, pode-se afirmar que a complexidade do cálculo do fluxo máximo é $O(NM)$ para cada nó avaliado, com N sendo o número de nós do grafo [9].

No grafo da figura 3.1, se s deseja enviar uma mensagem para t, poderá utilizar as arestas (s, a) e (s, b). Para a avaliação do fluxo máximo, o nó s será removido, conforme mostrado na figura 3.8, e os nós a e b serão avaliados. O fluxo máximo entre os nós a e t no novo grafo é igual a 2, portanto $c_1\big((s, a)\big) = 2$. O fluxo máximo entre os nós b e t no novo grafo é igual a 1, portanto $c_1\big((s, b)\big) = 1$.

Tendo em vista que o foco deste trabalho é o roteamento tolerante a falhas, o peso correspondente a esse critério (ω_1) deverá possuir um valor positivo e relativamente alto. Desta forma, o compromisso com a redundância será seguido. A discussão sobre o impacto de diferentes valores de pesos para esse critério é feita no capítulo 4, através da experimentação em um ambiente de simulação.

3.5.2 Critério c_2: Comprimento do Menor Caminho

Como forma de possibilitar a manutenção de um compromisso com a distância do caminho, um dos critérios utilizados na avaliação das arestas é a distância mínima entre o nó correspondente à aresta avaliada e o destino. Essa distância é definida como o comprimento do menor caminho (caminho com menor comprimento) que alcança o destino a partir do nó avaliado. Esse critério será chamado de c_2.

O cálculo desse critério pode ser feito através do algoritmo de Dijkstra [19], descrito na seção 2.1.1.2. Tendo em vista que as arestas não possuem pesos (todas as arestas são consideradas igualmente), a execução desse algoritmo se mostrará equivalente a uma busca em largura. Desta forma, utilizamos uma busca em largura, na qual o número de rodadas, ou níveis, percorridos para encontrar o nó de destino, partindo do nó avaliado, será utilizado como o valor do critério. A busca em largura é executada em $O(M)$, com M sendo o número de arestas da rede [9].

O cálculo desse critério pode ser ilustrado a partir do grafo da figura 3.1. Nesta figura, se s deseja enviar uma mensagem para t, poderá utilizar as arestas (s, a) e (s, b). Para a avaliação do fluxo máximo, o nó s será removido, conforme mostrado na figura 3.8, e os nós a e b serão avaliados. O menor caminho entre os nós a e t no novo grafo é o caminho (a, e, t), que possui tamanho 2, portanto $c_2\big((s, a)\big) = 2$. O menor caminho entre os nós a e t no novo grafo é o caminho (b, f, t), também com tamanho 2, portanto $c_2\big((s, b)\big) = 2$.

Para que seja possível escolher o caminho redundante com o menor comprimento, esse critério deverá ter peso negativo ($\omega_2 < 0$), visto que, quanto maior o comprimento do caminho mínimo, maior é o valor desse critério. Esse critério possui grande importância, visto que com grande freqüência, quando a rede estiver relativamente estável, o caminho

avaliado por esse critério será utilizado como rota final para o destino desejado.

3.6 Provas

Esta seção apresenta provas de correção do algoritmo, do número e tamanho das mensagens utilizadas, da complexidade do processamento e da latência de correção do algoritmo proposto.

3.6.1 Correção

O primeiro teorema a ser provado corresponde à correção do algoritmo, ou seja, se houver um caminho entre dois nós, o algoritmo efetuará o roteamento de uma mensagem com sucesso entre esses dois nós. Esse teorema assume algumas hipóteses para sua correção. A primeira hipótese tem relação com o conhecimento por parte dos nós vizinhos a uma aresta ou a outro nó do estado dessa aresta ou nó. Não existe diferenciação para um nó da rede se a falha está na aresta que liga esse nó a um outro nó da rede ou se a falha é nesse outro nó. Outros teoremas serão utilizados na seqüência desta seção para provar que, após um período de latência, os nós vizinhos possuem esta informação.

A segunda hipótese utilizada é a de que o nó de origem do roteamento precisa conhecer pelo menos um caminho não falho até o destino, ou seja, na representação local da topologia no nó de origem, pelo menos um dos caminhos disponíveis entre a origem e o destino no grafo não está falho na rede. Essa hipótese é suficiente (não necessária) para garantir a funcionalidade do algoritmo, porém não é necessário ao nó de origem saber qual dos caminhos conhecidos está sem falhas. Caso a origem não conheça nenhum caminho até o destino, nenhuma aresta será selecionada para a comunicação, e a mensagem não será enviada. O mesmo pode ocorrer se todos os caminhos conhecidos pela origem estiverem falhos.

Observe que a hipótese de que é necessário conhecer pelo menos um caminho sem falhas não é necessária para o funcionamento do algoritmo. Existem diversos casos em que os caminhos conhecidos pela origem podem estar falhos, mas a mensagem chega no destino

corretamente. Além disso, o caminho final utilizado não precisa necessariamente ser conhecido pela origem. Podem haver situações em que novas arestas ou arestas falhas que passam a funcionar são utilizadas no caminho final sem que seja necessário o conhecimento dessa aresta por parte do nó de origem do roteamento.

Na terceira hipótese assume-se que não haja mudança de topologia da rede entre o início do envio de uma mensagem até o momento em que a mensagem chega ao destino final, ou até o momento em que a origem toma conhecimento da inexistência de caminhos disponíveis ao destino. Essa hipótese é utilizada para garantir que, quando uma mensagem precisar retornar a um nó já visitado em virtude de falhas em caminhos conhecidos, a aresta utilizada na ida esteja também disponível na volta da mensagem.

Para a prova deste teorema, inicialmente é provado um lema que indica que, se a mensagem retornar à origem após o envio de mensagens por todas as arestas com caminho conhecido, a origem terá o conhecimento de que não há caminho conhecido sem falhas e ficará sem alternativas de roteamento. Em seguida é provado um lema que indica que, se houver um caminho funcional iniciando pela aresta escolhida para o roteamento, esse caminho será utilizado. Na seqüência, é provado um lema que indica que, se houver um ou mais caminhos conhecidos pela origem sem falhas, então uma aresta que pertença a um desses caminhos será selecionada para o roteamento. Finalmente, utilizando os lemas provados, é provado o teorema da correção do algoritmo.

Lema 3.1. *Considere $G = (V, E)$ um grafo, e $s, t \in V$ dois nós não falhos desse grafo. Considere que todos os nós possuem o conhecimento do estado de seus nós (ou arestas) vizinhos. Se s enviar uma mensagem que tenha destino t utilizando o algoritmo proposto neste trabalho, e todas as arestas avaliadas forem desprezadas ou, após o envio, retornarem a mensagem para a origem, então a origem s terá conhecimento de que não possui alternativas para o roteamento.*

Prova. Assume-se que o conjunto de arestas adjacentes e sem falhas a s seja o conjunto $\{e_1, e_2, \ldots e_n\}$. Nesse conjunto, são desprezadas arestas que não possuam caminhos conhecidos até t e que serão naturalmente desprezados pelo algoritmo. Como base de uma indução, é assumido que esse conjunto esteja vazio, ou seja, s não possui arestas adjacen-

tes. Neste caso, a prova é trivial, pois s não possui alternativas para o roteamento.

Como desenvolvimento da indução, é assumido como hipótese que o lema é verdadeiro para o conjunto de arestas $\{e_1, e_2, \ldots e_{n-1}\}$. Deve-se provar que o mesmo lema é verdadeiro caso o conjunto de arestas adjacentes à origem s seja o conjunto $\{e_1, e_2, \ldots e_{n-1}, e_n\}$. Assume-se, sem perda de generalidade, que a aresta escolhida para o roteamento é a aresta e_n. Portanto, a mensagem será enviada para o próximo nó utilizando essa aresta, nó que será chamado de u. Uma das hipóteses afirma que a mensagem retorna para a origem através dessa aresta. Pelo algoritmo, o retorno de uma mensagem ocorre apenas após o envio, do nó u para o nó s, de uma mensagem com todas as informações atualizadas de caminhos disponíveis utilizando o nó u. Esse retorno ocorre apenas quando o nó u não possui alternativas de roteamento até o destino t, e portanto não possui caminhos disponíveis até o destino t. Ao receber a mensagem de atualização, o nó s atualizará sua lista de roteamento. Com o retorno da mensagem original, o nó s fará uma nova avaliação de suas arestas, e a aresta e_n será então desprezada, visto que a informação mais atual recebida dessa aresta aponta a inexistência de caminhos até o destino t. Portanto, o conjunto de arestas disponíveis será $\{e_1, e_2, \ldots e_{n-1}\}$, para o qual o lema é verdadeiro. Logo, prova-se que s ficará sem alternativas para roteamento. □

Lema 3.2. *Considere $G = (V, E)$ um grafo, $s, t \in V$ dois nós não falhos desse grafo. Considere que todos os nós possuem o conhecimento do estado de seus nós (ou arestas) vizinhos. Considere que há um caminho de s até t, e que (s, u) é a primeira aresta desse caminho (u pode ser igual a t). Se s enviar uma mensagem para t através da aresta (s, u), utilizando o algoritmo proposto neste trabalho, a mensagem chegará ao destino t, ou retornará até a origem s.*

Prova. Como base de uma indução, assume-se que o grafo G possui 2 nós. Esses nós correspondem aos nós s e t que, por definição, são diferentes. A única aresta que pode ser avaliada pelo nó s é a aresta (s, t), visto não haver outra possibilidade de existência de arestas nesse grafo, logo $t = u$. Se essa aresta estiver falha, pela hipótese s tem conhecimento desta falha, e (s, u) não pode ser utilizado para o roteamento, caso em que a hipótese não se aplica. Se, porém, a aresta não estiver falha, então a mensagem será

enviada pelo nó s diretamente ao nó t pela aresta (s, t), chegando ao nó t e confirmando a base da indução.

Assumindo como hipótese de indução que o lema é válido para um grafo com $(n-1)$ nós, deve-se provar que o lema também é válido para n nós. Suponha, então, que G possui n nós. Quando s envia a mensagem usando a aresta (s, u), assume-se que s sabe que a aresta não está falha, visto que s possui conhecimento do estado das arestas vizinhas. O nó u, que recebe a mensagem, ao realizar a avaliação das arestas, remove o nó s do grafo utilizado para a avaliação, e portanto continua o roteamento em um grafo com $(n-1)$ nós, no qual o lema é válido. Logo, por indução, prova-se que o lema é válido para qualquer número de nós. □

Lema 3.3. *Considere $G = (V, E)$ um grafo, $s, t \in V$ dois nós não falhos deste grafo. Considere que todos os nós possuem o conhecimento do estado de seus nós (ou arestas) vizinhos. Considere que s conhece um caminho válido até o destino t. Se s deseja enviar uma mensagem para t, utilizando o algoritmo proposto neste trabalho, uma aresta que pertença a um caminho válido será selecionada.*

Prova. Como base de uma indução, considere que s possui apenas um vizinho, denominado de u. Como há um caminho válido de s até t, esse caminho precisa passar por u (que pode ser o próprio t). Na seleção das arestas para avaliação, a aresta (s, u) será considerada (visto que s conhece um caminho sem falhas que passa pela aresta), e como é a única aresta disponível, será selecionada para o roteamento. Pelo lema anterior, como a mensagem é enviada através dessa aresta, e a aresta pertence a um caminho válido, então a mensagem chega ao destino.

Em seguida, considerando que o lema é válido para $(n-1)$ vizinhos, considere que s possui n vizinhos, denominados de u_1, u_2, $\ldots u_{n-1}$, u_n. Pelo algoritmo, algumas arestas serão desprezadas por não pertencerem a caminhos válidos conhecidos até o destino. Suponha que a aresta (s, u_k) seja desprezada, qualquer que seja $1 \leq k \leq n$. Nesse caso, o algoritmo continua sem a aresta, como se o nó possuísse $(n-1)$ arestas, caso em que o lema é válido.

Considere que nenhuma aresta é desprezada. Nesse caso, sem perda de generalidade,

considere que a aresta com melhor avaliação é a aresta (s, u_n). O algoritmo envia uma mensagem por essa aresta. Se a aresta pertencer a um caminho válido até t, a mensagem chegará ao destino, conforme provado no lema 3.2. Se, porém, não houver caminho válido, pelo lema 3.1, a mensagem retorna ao nó u_n sem alternativas de roteamento, e este nó envia a mensagem de volta para o nó s, com informações mais atuais referentes ao estado das arestas da rede. A aresta (s, u_n) será, então, desprezada, por não pertence a nenhum caminho válido até o nó t. O algoritmo continua com $(n-1)$ arestas, caso em que o lema é válido. Com isto, provamos que o lema é válido para qualquer número de vizinhos da origem. □

Teorema 3.4. *Considere $G = (V, E)$ um grafo, e $s, t \in V$ dois nós não falhos desse grafo. Considere que os nós possuem o conhecimento da situação de seus nós vizinhos. Considere, também, que o nó s conhece pelo menos um caminho para o destino t que esteja funcional (sem falhas). Se s enviar uma mensagem para t utilizando o algoritmo proposto neste trabalho, a mensagem chegará até t.*

Prova. Como primeiro caso, considere a existência de uma aresta sem falha que liga diretamente s a t. Nesse caso, como s é vizinho à aresta e, portanto, possui conhecimento de sua funcionalidade, pelo algoritmo, a mensagem de s a t será enviada pela aresta $s-t$, chegando ao nó t, conforme proposto.

Como segundo caso, considere a não existência de uma aresta ligando diretamente os nós s e t. Esse segundo caso se aplica também quando a aresta ligando os dois nós existe e está falha. Pelo lema 3.3, como é assumido que a origem s conhece um caminho sem falhas até o destino t, logo foi concluído que uma aresta que possui um caminho sem falhas é escolhida. Pelo lema 3.2, quando a aresta escolhida possui um caminho sem falhas, a mensagem chega ao destino, logo, conclui-se que a mensagem chega ao destino t. □

3.6.2 Número e Tamanho das Mensagens de Atualização

Teorema 3.5. *Considere uma rede que utiliza, para o roteamento, o algoritmo proposto neste trabalho. O número de mensagens enviadas para atualização da topologia entre os*

nós é $O(M)$ a cada α segundos, e cada mensagem tem tamanho $O(M)$.

Prova. O algoritmo proposto neste trabalho utiliza mensagens periódicas com as mudanças de topologia ocorridas na rede. De acordo com o algoritmo descrito neste trabalho, cada um dos nós da rede envia uma mensagem a cada α segundos para cada um de seus vizinhos. Definindo g_v como o grau (ou número de arestas adjacentes) do nó v, cada nó enviará g_v mensagens a cada α segundos. Como, pela teoria dos grafos, $\sum_{v \in V} g_v = 2M$, então a cada α segundos, $2M$ mensagens serão enviadas. Logo, conclui-se que o número de mensagens enviadas é $O(M)$ a cada α segundos.

Para o tamanho da mensagem, o pior caso de uma mensagem será quando as informações referentes a todas as arestas forem enviadas em uma mensagem. Como cada informação de aresta é contada duas vezes (uma para cada nó adjacente à aresta), conclui-se que a mensagem com maior tamanho possível possui $2M$ informações de arestas, logo o tamanho da mensagem, no pior caso, é $O(M)$. □

3.6.3 Complexidade do Roteamento

Teorema 3.6. *Considere uma rede que utiliza, para o roteamento, o algoritmo proposto neste trabalho. O tempo necessário para escolha de uma rota entre uma origem e um destino é $O(M^3)$.*

Prova. Tendo em vista que o cálculo de uma rota é dinâmico, ou seja, cada nó calcula apenas a próxima aresta, assume-se que a complexidade do cálculo da rota é proporcional ao tamanho da rota, ou seja, ao número de nós visitados. Em cada um dos nós visitados, o algoritmo de escolha da aresta a ser utilizada é executado. Logo, pode-se concluir que a complexidade da escolha de uma rota será o número de nós visitados multiplicado pela complexidade da escolha de uma aresta.

No envio de uma mensagem através do algoritmo proposto, cada um dos nós poderá receber a mensagem, no pior caso, uma vez de cada uma de suas arestas. Tendo em vista as características do algoritmo, quando a mensagem é enviada através de uma aresta em um sentido específico, por exemplo, através da aresta (a, b), no sentido de a para b, ao

retornar para o nó a, esse nó não selecionará a mesma aresta novamente. Logo, podemos assumir que o tamanho da rota, ou o número de nós visitados pela mensagem, é $O(M)$.

Para a seleção de uma aresta, o algoritmo realiza o cálculo de $\Gamma(G, e)$ para cada uma das arestas vizinhas ao nó, ou seja, para g_v arestas. O cálculo de $\Gamma(G, e)$ envolve o cálculo de diversos critérios, que neste trabalho se resumem a dois: o critério c_1, correspondente ao fluxo máximo, que possui complexidade $O(NM)$; e o critério c_2, correspondente ao caminho de comprimento mínimo, que possui complexidade $O(M)$. Logo, a complexidade do cálculo de $\Gamma(G, e)$ é $O(NM + M)$, ou simplesmente $O(NM)$. Multiplicando esse resultado pelo número de arestas vizinhas, temos $O(NM\overline{g_v})$. Como $N\overline{g_v} = 2M$, o algoritmo é executado em $O(M^2)$.

Finalmente, para o roteamento completo de uma mensagem, conclui-se que o algoritmo executa, no pior caso, em $O(M^2M) = O(M^3)$. □

3.6.4 Latência

Para o cálculo da latência de funcionamento do algoritmo, são analisadas as hipóteses propostas para o funcionamento do algoritmo, com ênfase nas seguintes: o algoritmo assume que os vizinhos de uma aresta têm conhecimento da situação da mesma; o algoritmo também assume que a origem do roteamento possui conhecimento de pelo menos um caminho não falho até o destino. Tendo em vista estas hipóteses, calcula-se o tempo necessário para que, após uma alteração na topologia, o roteamento possa funcionar apropriadamente.

Teorema 3.7. *Considere uma rede que utiliza, para o roteamento, o algoritmo proposto neste trabalho. Considere que a origem de uma mensagem já conhece um caminho que está sem falhas até o destino. Considere que ocorreu um evento de alteração de estado em uma aresta da rede. O tempo máximo necessário a partir desse evento para que a mensagem chegue ao destino corretamente é de β segundos.*

Prova. Assume-se inicialmente o caso em que a origem já conhece um caminho sem falhas até o destino que não passa pela aresta que sofreu alteração de estado. Nesse

caso, só é necessário que o vizinho tome conhecimento da alteração de estado. Caso a alteração de estado corresponda à falha de uma aresta, cada vizinho tomará conhecimento da falha caso nenhuma mensagem seja recebida através da aresta em β segundos. Caso a alteração de estado seja a inclusão de uma nova aresta, ou de uma aresta falha que passa a funcionar, então o conhecimento da nova aresta se dará no recebimento de uma mensagem de atualização de topologia, que ocorre a cada α segundos. Como $\beta > \alpha$, assume-se que a latência é β segundos. □

Teorema 3.8. *Considere uma rede que utiliza, para o roteamento, o algoritmo proposto neste trabalho. Considere que a origem de uma mensagem não conhece um caminho que está sem falhas até o destino. Considere que ocorreu um evento de alteração de estado em uma aresta da rede. O tempo máximo necessário a partir deste evento para que a mensagem chegue ao destino corretamente é de $O(D(G)\alpha)$ segundos, sendo $D(G)$ o diâmetro do grafo correspondente à topologia da rede.*

Prova. Inicialmente, considere o caso da origem não conhecer nenhum caminho que chega ao destino. Entretanto, há um caminho que chega ao destino passando por uma aresta falha e passa a funcionar. Isto pode ocorrer quando a aresta que sofre alteração de estado particiona a rede, ou no início do funcionamento de uma rede, quando nenhum dos nós possui informações suficientes para roteamento. Para que os vizinhos da aresta tenham conhecimento da alteração de situação, são necessários α segundos. Essa informação deverá ser enviada para os nós vizinhos dos vizinhos, e assim sucessivamente até que o nó que realizará o roteamento tenha a informação. Em cada nó visitado pela informação de estado da aresta, a informação poderá levar α segundos para ser enviada ao próximo vizinho. Como cada nó envia a mensagem para todos os seus vizinhos, a mensagem chegará ao nó que necessita da informação no número de rodadas correspondente à distância mínima entre este nó e os nós vizinhos da aresta citada. Como esta distância é no máximo igual ao diâmetro do grafo, ou $D(G)$, concluímos que, para este caso específico, a latência de roteamento será $O(D(G)\alpha)$. □

Capítulo 4

Implementação e Resultados Experimentais

Tendo em vista a visualização e a experimentação do algoritmo de roteamento proposto neste trabalho, foi desenvolvida uma implementação do algoritmo. Este capítulo descreve essa implementação, bem como os resultados obtidos através da execução da implementação.

O restante do capítulo está organizado como segue. A seção 4.1 descreve cada um dos módulos da implementação desenvolvida. A seção 4.2 descreve os resultados obtidos na execução do processo em ambiente de simulação. Finalmente, a seção 4.3 descreve as conclusões obtidas pela execução do algoritmo.

4.1 Descrição da Implementação

A implementação do algoritmo proposto foi feita utilizando a linguagem Java [65], na versão J2SE (*Java 2 Standard Edition*) 1.4.2. A implementação foi separada em 3 módulos: um módulo interno, responsável pelo funcionamento básico do algoritmo, como a comunicação entre nós e arestas, a escolha das arestas para roteamento e o envio e recebimento de mensagens de atualização de rotas; um módulo para interface gráfica, utilizado para visualização do funcionamento do algoritmo através de interações com o usuário; e um módulo para experimentação, utilizado para geração dos resultados de simulação do algo-

ritmo em redes aleatórias com situações aleatórias. Cada um dos módulos é descrito de forma geral na seqüência. A implementação está disponível para execução na Web, em `http://web.inf.ufpr.br/jonatan/mfrp`.

4.1.1 Módulo Interno: Algoritmo de Roteamento

O principal módulo da implementação é o módulo interno, responsável pelo funcionamento básico do algoritmo de roteamento proposto neste trabalho. Esse módulo está organizado em diversas classes, conforme descrito a seguir.

A primeira classe disponível no módulo interno é a classe `Node`. Essa classe é responsável pelo funcionamento interno de um nó da rede, incluindo o armazenamento de informações como o controle de funcionamento do nó, a fila de mensagens a processar, a topologia conhecida da rede e as informações enviadas e recebidas a respeito dos demais nós da rede. A classe `Node` implementa também métodos utilizados para realizar o roteamento, com funções como o envio periódico de mensagens para os nós vizinhos, o controle da situação dos enlaces vizinhos, o recebimento e tratamento de mensagens e a escolha de enlaces para roteamento.

Cada objeto `Node` possui associado ao mesmo uma `Thread`, que refere-se a uma instância de execução paralela da linguagem Java. Exceto pela inclusão de mensagens na pilha de mensagens, que é realizada pelo enlace adjacente ao nó, todas as demais operações são executadas exclusivamente pela `Thread` associado ao objeto `Node`. Essa `Thread` permanece executando em *background* durante o tempo de vida da aplicação.

Outra classe do módulo interno é a classe `Link`, responsável pelo funcionamento de um enlace da rede. Essa classe possui, além das informações específicas de um enlace, como os nós adjacentes ao mesmo e o estado, operações para realizar o envio de uma mensagem através do enlace. Existe um tratamento para criar um atraso parametrizável no envio da mensagem através de um enlace, porém esse tratamento foi transferido para a classe `Node` para evitar a criação de um objeto `Timer` (e, por conseqüência, mais uma `Thread`) na classe `Link`, evitando assim o uso desnecessário de memória e tempo de processamento.

O módulo interno também possui uma classe, denominada `Parameters`, contendo as

informações parametrizáveis do algoritmo. Essa classe é utilizada pela classe `Node` para obtenção dos valores associados a cada parâmetro. Os valores disponíveis nessa classe são: os pesos dos critérios de avaliação de arestas (fluxo máximo e caminho mais curto), o tempo entre o envio de uma mensagem de atualização e a próxima (α), o atraso (*timeout*) após o recebimento de uma mensagem de atualização em que a aresta é considerada falha (β), e o tempo de atraso no envio de uma mensagem em cada aresta.

A interface `Debug` é utilizada pelos módulos que utilizam o módulo interno para receber informações úteis do mesmo, como o recebimento de uma mensagem por um nó, a alteração do estado de uma aresta, entre outros. Os métodos da interface podem ser implementados pelas classes que utilizam o módulo interno, com o código a ser executado no momento da ocorrência dos eventos tratados pela classe. A classe `NoDebug` é uma classe auxiliar que implementa essa interface, utilizada caso não haja intenção de conhecer a ocorrência dos eventos citados.

A classe `Digraph` é utilizada para armazenar uma estrutura de grafos direcionados, por exemplo na representação local da rede existente em cada nó da rede. Os nós de um grafo podem ser objetos de qualquer classe, enquanto as arestas são instâncias da classe `Edge`, também definida no módulo. A classe possui métodos para cálculos úteis em grafos, como busca em largura e cálculo do fluxo máximo. A classe `Path` é utilizada internamente á classe `Digraph` para representar um caminho no grafo.

O envio de mensagens deve ser feito utilizando a classe abstrata `Message`. Essa classe possui informações básicas comuns a todas as mensagens do algoritmo, como origem, destino, uma lista dos nós visitados e o próximo nó a ser visitado. Qualquer tipo específico de mensagem deve ser tratado como uma subclasse dessa classe.

Dois tipos específicos de mensagem são tratados no módulo interno. O primeiro refere-se a mensagens de atualização entre nós vizinhos, tratadas através da classe `NeighbourUpdateMessage` que possui informações como uma lista de alterações de estado de arestas da rede. Cada registro de alteração de estado é tratado na classe `EdgeInformation`, que possui como propriedades uma aresta, um estado e um controle de ordenação (*timestamp*).

Outro tipo específico de mensagem tratado no módulo interno é a confirmação de

recebimento de mensagens, tratada através da classe `AckMessage`. Essa classe possui apenas uma indicação da mensagem original.

4.1.2 Módulo de Interface Gráfica

A implementação desenvolvida para o algoritmo proposto neste trabalho possui também a implementação de uma interface gráfica, utilizada para visualização do funcionamento do algoritmo. Uma versão dessa interface gráfica está disponível em `http://web.inf.ufpr.br/jonatan/mfrp`, e pode ser vista na figura 4.1.

A interface gráfica desenvolvida para o algoritmo é baseada na estrutura de applets Java [66], utilizando a interface gráfica Swing [67]. A interface foi separada em três partes, acessíveis através de guias: uma contendo a visualização e manipulação da rede e dos roteamentos desejados, uma para depuração de resultados e uma para visualização e alteração dos valores dos parâmetros do algoritmo.

A primeira guia da interface gráfica contém um menu à esquerda com opções para seleção da ação desejada, e uma área à direita para realização das ações. Nessa tela, é possível manipular a topologia e o estado dos nós e arestas do grafo correspondente à rede ou solicitar o roteamento de uma mensagem.

Para criar um nó no grafo, é necessário selecionar a opção *New Nodes* e clicar na posição em que o nó deve ser posicionado. Para alterar a posição de um nó do grafo, é necessário selecionar a opção *Move Nodes* e arrastar, com o ponteiro do mouse, o nó desejado para a nova posição. Para remover um nó do grafo, é necessário selecionar a opção *Remove Nodes* e clicar sobre o nó que deve ser excluído.

Para criar uma aresta no grafo, é necessário selecionar a opção *New Edges*, clicar sobre o nó de origem da aresta, manter o mouse pressionado e movê-lo até o nó destino da aresta. Uma aresta criada em um sentido valerá para ambos os sentidos. Para remover uma aresta, é necessário selecionar a opção *Remove Edges* e clicar sobre a aresta que deve ser excluída.

Para alterar o estado de um nó ou aresta, deve-se selecionar, respectivamente, a opção *Toggle Node State* ou *Toggle Edge State* e clicar sobre o nó ou aresta a ser alterado. Se

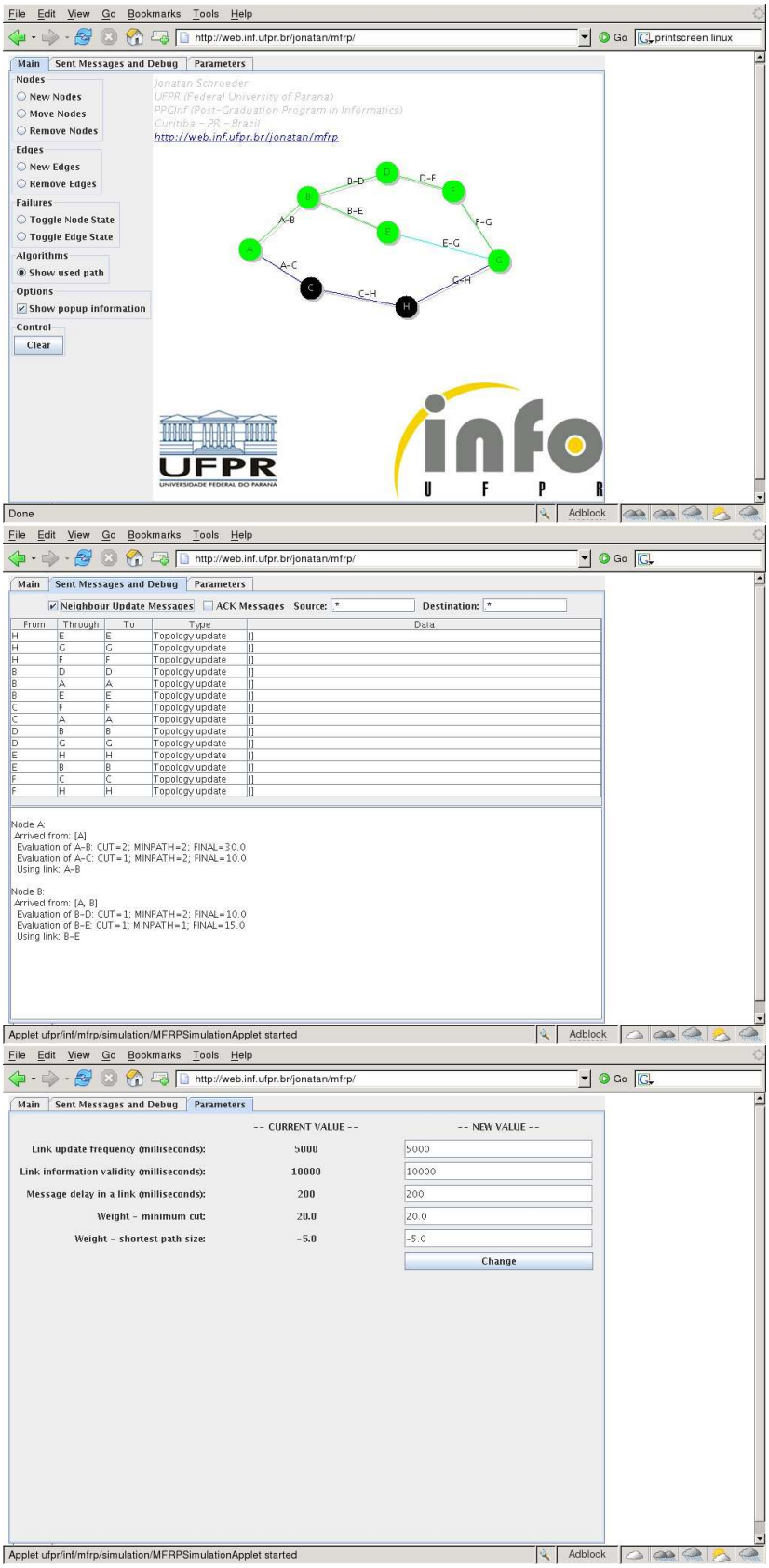

Figura 4.1: Um exemplo de utilização da interface gráfica.

o nó ou aresta estiver funcional, ele será considerado falho, e vice-versa. Um nó falho é apresentado como vazado, enquanto que uma aresta falha é apresentado em uma cor mais clara. A região de uma aresta mais próxima a um nó mantém a informação do estado da aresta na topologia local do nó. Desta forma, é possível visualizar se os nós vizinhos a uma aresta já possuem a informação mais atual de seu estado.

Para realizar o roteamento de uma mensagem, é necessário selecionar a opção *Show Used Path*, clicar sobre o nó de origem da mensagem, manter o mouse pressionado e movê-lo até o nó destino da mensagem. Os nós e arestas pelos quais a mensagem passa são destacados, de forma que é possível visualizar o caminho utilizado para o roteamento.

Caso o ponteiro do mouse seja mantido sobre um nó ou aresta, são mostradas informações relativas ao estado e aos dados disponíveis no nó ou aresta. Para um nó, são exibidos a identificação do nó, o estado atual, os vizinhos conhecidos pelo nó, os vizinhos inativos do nó (vizinhos que são considerados falhos, mas para os quais é necessário continuar enviando mensagens, de forma que quando a aresta correspondente voltar a funcionar, uma mensagem seja enviada) e as arestas conhecidas pelo nó. Para uma aresta, são exibidos os nós adjacentes à aresta, o estado da aresta e a informação que os nós adjacentes possuem a respeito da aresta. Para que essas informações não sejam mostradas, deve-se desmarcar a opção *Show popup information*.

O botão *Clear*, disponível nas opções, permite que todos os nós da rede sejam removidos, para que uma nova rede seja criada.

A segunda guia da interface gráfica é dividida em duas áreas, uma superior e uma inferior. A área superior exibe as últimas mensagens transmitidas na rede. Há uma opção para mostrar ou não mensagens de atualização e mensagens de confirmação. Há também duas caixas de texto para filtrar as mensagens enviadas pela origem ou pelo destino. Essas caixas de texto devem ser preenchidas com expressões regulares, com as quais o nome dos nós de origem e destino da mensagem será comparado. A área inferior da guia é utilizada para informações sobre o roteamento realizado. Nessa área, são listadas informações como os nós visitados por uma mensagem e os resultados das avaliações das arestas para o roteamento.

A terceira guia da interface gráfica possui uma listagem dos parâmetros utilizados pelo algoritmo. Esses parâmetros são os mesmos parâmetros descritos na seção 4.1.1 para a classe `Parameters`. A tela permite a visualização dos valores correntes utilizados pelo algoritmo, possibilitando a alteração de qualquer um dos valores. Algumas alterações de parâmetros, como a do tempo de espera entre duas mensagens de atualização, podem levar algum tempo para serem aplicadas, visto que pode ter sido iniciado um processo que utiliza o valor informado.

4.1.3 Módulo de Simulação

Para possibilitar a simulação do algoritmo em redes de grande porte, foi criado um módulo específico para execução do algoritmo proposto em redes aleatórias, com eventos aleatórios. Esse módulo possui interface em modo texto, e utiliza o módulo interno para execução do algoritmo. Todos os processos desenvolvidos nesse módulo têm os parâmetros passados na forma de argumentos, na chamada dos processos.

No módulo de simulação, os processos de geração de grafos, geração de eventos e execução do processo foram separados, de forma a possibilitar que o comportamento do algoritmo com diferentes entradas seja analisado com os mesmos grafos, e desta forma permitir uma confiabilidade dos resultados.

O primeiro processo desenvolvido é o processo de geração de grafos aleatórios. Esse processo recebe como parâmetros o tamanho (número de nós) do grafo, o tamanho do *backbone* inicial (m_0), a probabilidade de criação de novo nó em cada iteração (p) e um parâmetro para seleção de nós aleatórios (β). A partir desses parâmetros, uma rede é gerada com topologia *Power Law* [68], topologia esta que é comprovadamente similar à da Internet [69, 70, 71]. O algoritmo utilizado para geração de grafos aleatórios é o apresentado por Bu e Tosley [72]. O processo gera um arquivo texto contendo a descrição da topologia da seguinte forma: para cada nó, uma linha contendo o texto "NODE", seguido de um espaço e do nome do nó; para cada aresta, uma linha contendo o texto "EDGE", seguido de um espaço e dos nomes dos nós adjacentes à aresta.

Outro processo desenvolvido é utilizado para geração de eventos aleatórios para si-

mulação do algoritmo. Os eventos gerados são: falha ou recuperação de nó, falha ou recuperação de aresta e envio de mensagem. Esse processo recebe como parâmetros o arquivo gerado pelo processo anterior, contendo a topologia do grafo, o período estimado da simulação (ou tempo de término da mesma), o atraso inicial para início do envio de mensagens, e o tempo médio entre dois eventos de cada tipo de evento. Todos os tempos são apresentados em milisegundos. O atraso inicial das mensagens foi introduzido para evitar que mensagens sejam enviadas antes que os nós possuam o conhecimento inicial da topologia da rede. Os eventos são gerados com intervalo aleatório, com o intervalo médio equivalente ao tempo médio informado como parâmetro. São gerados eventos suficientes para completar o tempo de simulação estipulado nos parâmetros. O processo gera um arquivo texto contendo os eventos gerados da seguinte forma: para cada evento de alteração de estado de nó, uma linha contendo o texto "NODE_STATE", seguido do tempo (em milisegundos, a partir do início da simulação) em que ocorre o evento, seguido do nome do nó que possui o estado alterado; para cada evento de alteração de estado de aresta, uma linha contendo o texto "EDGE_STATE", seguido do tempo em que ocorre o evento, seguido do nome dos nós adjacentes à aresta que possui o estado alterado; para cada evento de envio de mensagem, uma linha contendo o texto "MESSAGE", seguido do tempo em que ocorre o evento, seguido do nome dos nós de origem e de destino da mensagem.

Finalmente, um processo foi criado para realizar a simulação do algoritmo de acordo com os eventos gerados no processo anterior. O processo de simulação recebe como parâmetros o nome dos arquivos de topologia e de eventos, além dos parâmetros do algoritmo descritos no capítulo 3 e na seção 4.1.1 (ω_1, ω_2, α, β e o atraso em cada aresta). A partir da topologia fornecida, o processo de simulação monta uma rede utilizando as classes descritas na seção 4.1.1, e executa os eventos informados como parâmetro nos tempos associados. Após o término da execução da simulação, um resumo contendo os resultados obtidos na simulação é apresentado, conforme ilustrado na figura 4.2.

O resumo apresentado no final do processo de simulação apresenta uma contagem das mensagens enviadas, das efetivamente recebidas e das que foram perdidas na execução do

```
Simulation begun
Time 874: EDGE STATE CHANGE for edge N17-N4 (failure)
Time 6190: EDGE STATE CHANGE for edge N22-N53 (failure)
Time 9622: EDGE STATE CHANGE for edge N14-N82 (failure)
Time 11951: EDGE STATE CHANGE for edge N1-N16 (failure)
Time 16751: EDGE STATE CHANGE for edge N4-N8 (failure)
Time 19447: EDGE STATE CHANGE for edge N22-N53 (working)
Time 20195: EDGE STATE CHANGE for edge N4-N84 (failure)
Time 21183: MESSAGE SENT from N30 to N26 (distance=2)
Time 21815: MESSAGE RECEIVED by N26 from N30 (2 visited nodes)
Time 21835: EDGE STATE CHANGE for edge N1-N22 (failure)
Time 22223: NODE STATE CHANGE for node N66 (failure)
Time 23007: MESSAGE SENT from N22 to N21 (distance=1)
Time 23415: MESSAGE RECEIVED by N21 from N22 (1 visited nodes)
...
Time 59613: MESSAGE SENT from N95 to N19 (distance=3)
Time 60449: MESSAGE RECEIVED by N19 from N95 (3 visited nodes)
RESULTS:
   Messages Sent:
      Total.................: 21
      Properly received......: 20 (95%)
      Source not working.....: 0 (0%)
      Destination not working: 0 (0%)
      No path available......: 1 (4%)
      Lost messages..........: 0 (0%)
   Resulting distance:
      Average path distance..: 3.15
      Average distance used..: 3.15
      Average path increase..: 0
```

Figura 4.2: Um exemplo de saída do processo de simulação do algoritmo proposto.

algoritmo. Mensagens em que o nó de origem ou de destino estavam falhos no momento do envio da mensagem são apresentadas em separado, assim como mensagens sem caminho disponível, visto que essas mensagens não poderiam ser roteadas normalmente nas condições em que se encontravam. Também são apresentados a distância média entre os nós de origem e destino das mensagens, assim como o número total de nós percorridos em média por cada mensagem.

4.2 Resultados Experimentais

Esta seção descreve resultados obtidos a partir da execução da simulação do algoritmo proposto neste trabalho. Inicialmente é comparado o funcionamento do algoritmo para diferentes valores para os parâmetros de tempo (α e β). Na seqüência, é feita uma avaliação da variação nos resultados com a utilização de valores diversos para os pesos dos critérios utilizados. Finalmente, é mostrado o comportamento do algoritmo em caso de uma seqüência de mensagens entre o mesmo par de nós, como ocorre em uma conexão que utiliza TCP.

Em todas as situações de simulação, é avaliada a variação do número de mensagens que efetivamente chegaram no destino. Para as simulações foram gerados 7 grafos, denominados G_0, G_1, G_2, G_3, G_4, G_5 e G_6. Para cada grafo foram gerados dois conjuntos de eventos, um para os testes de variação de α e β e dos pesos dos critérios, e um para avaliação em um fluxo de pacotes TCP, este com mensagens entre o mesmo par de nós em todas as mensagens.

Todos os grafos foram gerados com 100 nós, com m_0 igual a 5, p igual a 0,6 e β igual a 0,2. O número de nós utilizado foi limitado pelo espaço de memória disponível para execução do algoritmo, tendo em vista que cada nó da rede executa uma `Thread`, que utiliza grande espaço de memória. O conjunto de eventos gerado totalizou 10 minutos, com uma média de um evento de alteração de estado de nó a cada 120 segundos, um evento de alteração de estado de aresta a cada 60 segundos e uma mensagem a cada 2 segundos. Em todos os testes, a simulação foi realizada três vezes para cada conjunto de parâmetros e cada grafo, assumindo como resultado final o melhor entre os resultados

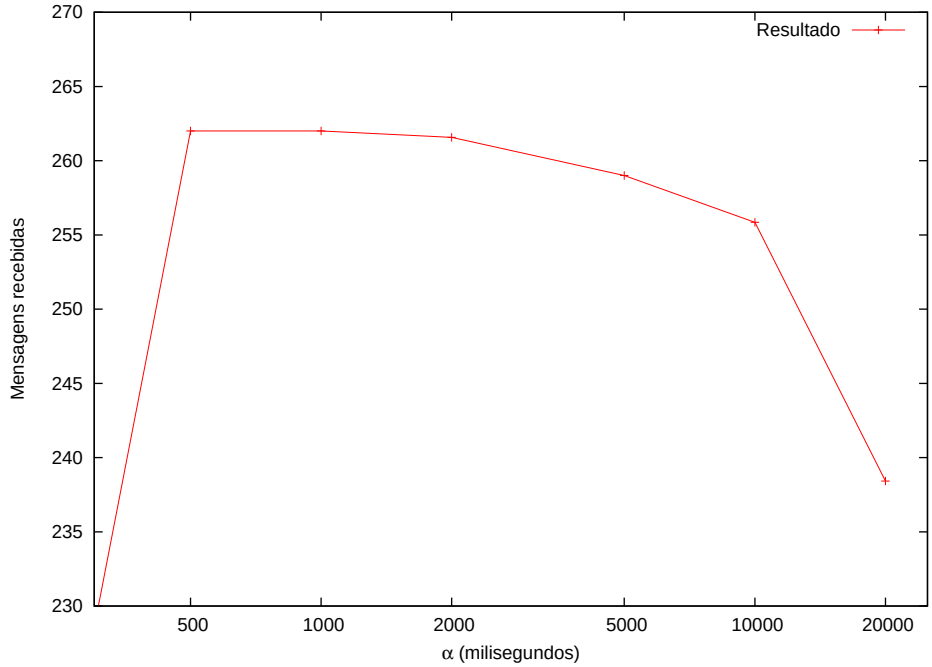

Figura 4.3: Comportamento do algoritmo conforme variação do valor do parâmetro α.

obtidos para cada situação.

Tendo em vista que o mesmo conjunto de eventos foi utilizado em todos os casos, as mensagens que não chegaram ao destino por falha na origem ou no destino, ou sem caminho funcional disponível, foram ignoradas, pois são idênticas em todos os casos. No total, as simulações do algoritmo geraram 286 mensagens em cada situação, em média, das quais 25 mensagens em média foram desconsideradas por um dos motivos citados.

4.2.1 Estudo da Variação dos Tempos Parametrizáveis

A simulação foi realizada com uma variação no tempo entre mensagens de atualização (conhecido por α). A simulação foi realizada com os seguintes valores para α: 10 milissegundos, 0,5 segundos, 1 segundo, 2 segundos, 5 segundos, 10 segundos e 20 segundos. Em todas as simulações, β foi definido como 2α, o *delay* foi definido como 0,2 segundos e foram utilizados os pesos 20 e -5 para os critérios c_1 e c_2. O resultado é ilustrado na figura 4.3. Nessa figura, os resultados referem-se à média da quantidade das mensagens que foram recebidas com sucesso pelo destino em todos os grafos.

Observando os resultados da simulação proposta, pode-se verificar que o desempenho do algoritmo é inferior para valores muito altos de α. O mesmo ocorre para valores

muito baixos de α, como 10 milissegundos, que por obter um resultado muito baixo não foi incluído no grafo, apenas cerca de 5 mensagens foram roteadas com sucesso neste caso. Isto pode ser explicado pelo fato de que, com um valor de α baixo, o número de mensagens trafegando na rede é maior, o que pode trazer congestionamento em alguns pontos da rede e atrasando as informações relativas a atualizações de rotas nos nós. Com o aumento do valor de α, há uma redução no número de mensagens na rede e, por conseqüência, as mensagens podem trafegar livremente. Porém à medida em que o valor de α cresce, cresce também o tempo necessário para que os nós vizinhos de uma aresta tenham conhecimento de uma alteração de estado e, desta forma, os nós passam a ter informações desatualizadas da topologia da rede por um tempo maior. Desta forma, para o conjunto de eventos citado, assume-se que o valor de α ideal estaria entre 0,5 segundo e 2 segundos, valor para o qual o resultado é melhor.

Também foi realizado um estudo da variação do tempo de *timeout* da informação de uma aresta (β). A simulação foi realizada com α fixado em 5 segundos, e com os seguintes valores para β: 5 segundos, 7,5 segundos, 10 segundos, 15 segundos, 20 segundos, 30 segundos e 50 segundos. Em todas as simulações, o *delay* foi definido como 0,2 segundos e foram utilizados os pesos 20 e -5 para os critérios c_1 e c_2. O resultado pode ser observado na figura 4.4. Nessa figura, os resultados referem-se à média da quantidade das mensagens que foram recebidas com sucesso pelo destino em todos os grafos.

A partir dos resultados obtidos com a variação do valor de β, identificamos que não há grande variação no resultado para a alteração do valor de β, porém o resultado é melhor para valores inferiores de β, mais próximos de α. Para valores maiores, a latência do algoritmo aumenta, aumentando também o tempo necessário para que nós vizinhos a uma aresta que falha tenham conhecimento dessa falha, e aumentando a probabilidade de envio de uma mensagem através de uma aresta falha.

4.2.2 Estudo da Variação dos Pesos dos Critérios

Visando avaliar a escolha dos pesos dos critérios, foram feitos testes com diversos conjuntos de pesos. Para tal, foi fixado um valor para um dos pesos, variando apenas o outro

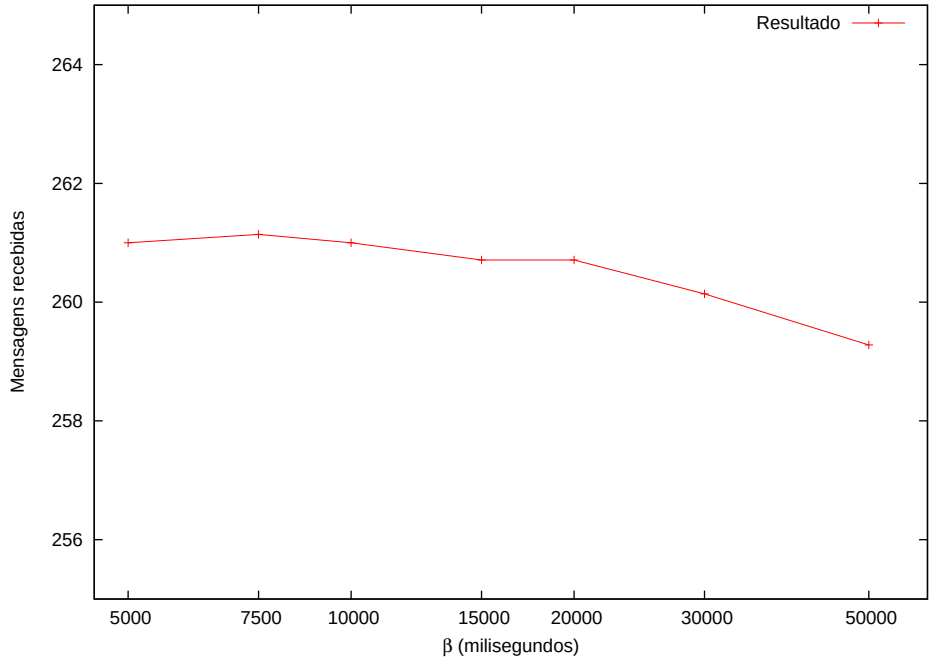

Figura 4.4: Comportamento do algoritmo conforme variação do valor do parâmetro β.

peso, tendo em vista que valores de pesos com mesma razão geram o mesmo resultado – independente dos valores de fluxo máximo e comprimento mínimo, pesos (ω_1, ω_2) iguais a $(20, -5)$ produzem exatamente os mesmos resultados que os pesos $(4, -1)$, visto que a aresta com maior avaliação no primeiro caso será sempre a mesma aresta que a do segundo caso.

A simulação foi realizada com os seguintes pares de valores de critérios (ω_1, ω_2): $(1, -5)$, $(2, -5)$, $(5, -5)$, $(7, -5)$, $(10, -5)$, $(15, -5)$, $(20, -5)$, $(30, -5)$, $(50, -5)$, $(0, -5)$ e $(1, 0)$. Em todas as simulações, o *delay* foi definido como 0,2 segundos, e aos parâmetros α e β foram atribuídos os tempos de 5 segundos e 10 segundos, respectivamente. O resultado pode ser observado na figura 4.5. Nessa figura, os resultados de (A) referem-se à média da quantidade das mensagens que foram recebidas com sucesso pelo destino em todos os grafos, enquanto (B) ilustra o aumento médio do tamanho do caminho utilizado em relação ao caminho mínimo disponível para o par de nós que troca mensagens.

Observando os resultados da simulação, pode-se verificar que, com a freqüência de eventos utilizada, não há grande variação no número de mensagens que chegam ao destino. Conclui-se, então, que os pesos utilizados não possuem grande influência nesse resultado. Comparando, porém, o tamanho das mensagens que chegam ao destino, ou

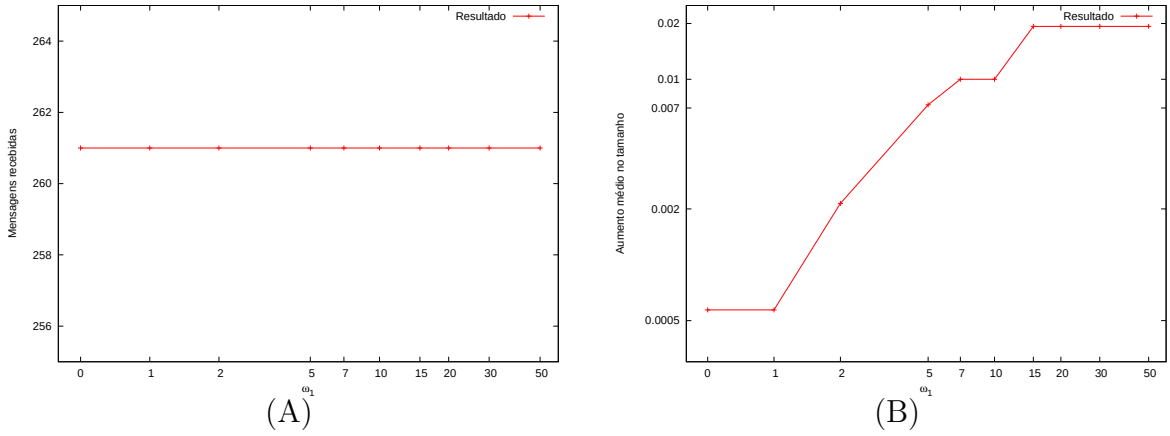

Figura 4.5: Comportamento do algoritmo conforme variação dos valores dos pesos dos critérios.

seja, o número de nós visitados pela mensagem, pode-se verificar que houve um aumento no tamanho da rota utilizada à medida em que o valor do peso do critério de fluxo máximo aumenta e, por conseqüência, a influência do critério de comprimento mínimo diminui. Esse comportamento é até certo ponto esperado, visto que, com um peso relativamente menor para o critério de comprimento mínimo, o caminho é escolhido dando maior ênfase para caminhos com comprimento maior, mas com fluxo máximo maior, e com isto aumentando a rota utilizada. O aumento no tamanho da rota, porém, é relativamente pequeno, com um aumento de 0,0193 em média para ω_1 igual a 50, valor considerado aceitável.

Para os critérios (ω_1, ω_2) iguais a $(1, 0)$, porém, o resultado não se mostrou interessante, gerando um aumento médio no tamanho do caminho de 5,6 arestas, ou seja, os caminhos gerados foram em média maiores que o dobro do caminho mínimo disponível entre os nós, que variou entre 3 e 3,3 conforme o grafo e conjunto de eventos utilizado.

4.2.3 Estudo do Fluxo de uma Conexão TCP sob Falha

Considerando que muitas aplicações que necessitam de conexões confiáveis por um período de tempo muito grande, onde cada conexão é realizada com um mesmo par de nós, foi realizada uma avaliação do funcionamento do algoritmo para um grande número de mensagens entre o mesmo par de nós. Foram utilizados os mesmos grafos descritos nas seções anteriores. O conjunto de eventos foi gerado com os mesmos parâmetros de freqüência de

eventos, porém apenas um par de nós foi utilizado.

Foi realizada uma simulação utilizando o processo descrito na seção 4.1, utilizando como tempos de *timeout* 5 e 10 segundos respectivamente para α e β, 0,2 segundos para o atraso em cada aresta, e os parâmetros 20 e -5 respectivamente para ω_1 e ω_2.

Como o conjunto de eventos gerado para o grafo G_3 foi o único em que houve alteração no caminho entre os nós utilizados para o fluxo de pacotes, esse conjunto foi selecionado para análise. A partir do resultado da simulação, foi possível identificar que, inicialmente, os nós escolhidos aleatoriamente para a comunicação possuíam uma distância de 3 arestas entre os mesmos. Durante a simulação, um dos nós do caminho utilizado inicialmente ficou falho. 8 mensagens enviadas logo na seqüência desse evento não chegaram ao destino. Na seqüência, uma mensagem foi enviada e, com idas e vindas por nós da rede, visitou 11 nós, mas chegou ao destino desejado. A mensagem seguinte, enviada poucos milissegundos após a anterior, também visitou diversos nós (19 ao todo), também chegando ao destino. Finalmente, a mensagem seguinte encontrou outro caminho com 3 arestas e, a partir deste momento, utilizou esse caminho para o roteamento.

Durante o tempo em que os nós realizaram a comunicação, foram totalizadas 285 mensagens recebidas com êxito pelo nó de destino, com apenas 8 mensagens sendo perdidas no roteamento, resultado considerado satisfatório.

4.3 Conclusão

Observando os resultados gerados, comprova-se que um número de mensagens superior a 90% das mensagens atingiu o destino. Observa-se que os melhores resultados para o conjunto de eventos utilizado foram obtidos para α próximo de um segundo, com queda no desempenho do algoritmo para valores muito maiores ou menores que este. Não houve grande variação no desempenho para a variação de β, porém os resultados mais satisfatórios são encontrados em valores inferiores deste parâmetro. A variação dos pesos dos critérios encontrou resultado aceitável, e o estudo do fluxo de mensagens de mesmo par de nós gerou bons resultados.

Capítulo 5

Conclusão

Neste trabalho foi apresentada uma proposta de roteamento tolerante a falhas, fundamentada em uma escolha dinâmica de caminhos, selecionados com base em uma avaliação de fluxo máximo, utilizando como critério secundário a distância dos caminhos. Foram inicialmente apresentados diversos conceitos relacionados ao roteamento em redes de computadores. A abordagem de roteamento proposta foi formalmente especificada. A correção do algoritmo foi provada em situações definidas, e concluiu-se que a complexidade tanto do roteamento quanto das mensagens de atualização é polinomial em termos do número de nós e arestas da rede. Foi obtida também a latência de convergência do roteamento proposto. Resultados experimentais obtidos através de simulação em redes com topologias similares às da Internet foram apresentados, utilizando diferentes valores para os parâmetros α, β, ω_1 e ω_2, possibilitando uma análise do funcionamento e da parametrização do algoritmo proposto. Também foi apresentada uma análise do funcionamento do algoritmo para um fluxo de mensagens similar a uma conexão TCP na Internet.

Trabalhos futuros incluem a avaliação de outros critérios para escolha de arestas para roteamento, incluindo critérios de QoS. Pretende-se, também, avaliar a funcionalidade do protocolo para redes móveis. Futuramente pretende-se formalizar um protocolo utilizando o algoritmo proposto neste trabalho, através da descrição das mensagens para uso em uma rede real.

Referências Bibliográficas

[1] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *SIGCOMM*, pages 175–187, 2000.

[2] Y. Rekhter and T. Li. *RFC 1771: A Border Gateway Protocol 4 (BGP-4)*, March 1995.

[3] J. Chandrashekar, Z. Duan, Z. L. Zhang, and J. Krasky. Limiting path exploration in BGP. disponível em `http://www-users.cs.umn.edu/~jaideepc/papers/epic-tr.pdf`.

[4] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang. Improving BGP convergence through consistency assertions. In *INFOCOM*, New York, 2002.

[5] L. Wang, D. Massey, K. Patel, and L. Zhang. FRTR: A scalable mechanism for global routing table consistency. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2004)*, pages 465–474, Florence, Italy, 2004.

[6] E. P. Duarte Jr., R. Santini, and J. Cohen. Delivering packets during the routing convergence latency interval through highly connected detours. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2004)*, pages 495–504, Florence, Italy, 2004.

[7] R. Santini, E. P. Duarte Jr., J. Schroeder, P. R. Torres Jr., and J. Cohen. Roteamento tolerante a falhas baseado em desvios de alta conectividade. Technical report, Universidade Federal do Paraná, 2004.

[8] L. R. Ford Jr. and D. R. Fulkerson. Flows in networks. *Princeton University Press*, 1962.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, second edition, 1990.

[10] Y. Rekhter and P. Gross. *RFC 1772: Application of the Border Gateway Protocol in the Internet*, March 1995.

[11] A. S. Tanenbaum. *Redes de Computadores*. Editora Campus, Rio de Janeiro, 3^{rd} edition, 1997.

[12] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[13] L. G. Roberts. The Arpanet and Computer Networks. In *Proceedings of the ACM Conference on The history of personal workstations*, pages 51–58, Palo Alto, 1986.

[14] L. Pouzin. *The Cyclades Computer Network: Towards Layered Network Architectures*. North-Holland Publishing Company, New York, 1982.

[15] C. Hedrick. *RFC 1058: Routing Information Protocol*, June 1988.

[16] C. Huitema. *Routing in the Internet*. Prentice Hall, Upper Saddle River, 2^{nd} edition, 1999.

[17] J. M. McQuillan, I. Richer, and E. C. Rosen. Arpanet Routing Algorithm Improvements. *BBN Technical Report 3803*, April 1978.

[18] J. M. McQuillan et al. The New Routing Algorithm for the Arpanet. *IEEE Transactions on Communications*, May 1980.

[19] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[20] G. Malkin. *RFC 2453: RIP Version 2*, November 1998.

[21] G. Malkin. *RFC 1721: RIP Version 2 Protocol Analysis*, November 1994.

[22] G. Malkin and F. Baker. *RFC 1389: RIP Version 2 MIB Extension*, January 1993.

[23] G. Malkin and R. Minnear. *RFC 2080: RIPng for IPv6*, January 1997.

[24] J. Moy. *RFC 1583: OSPF Version 2*, March 1994.

[25] F. Ross. An Overview of FDDI: The Fiber Distributed Data Interface. *IEEE Journal on Selected Areas in Communication*, 7(7), 1989.

[26] A. Grant and D. Hutchison. X25 Protocols and Local Area Networks. *Computer Networks: The International Journal of Distributed Informatique*, 6(4):255–262, September 1982.

[27] D. Mills. *Exterior Gateway Protocol (EGP) Formal Specification*, March 1982.

[28] K. Lougheed and Y. Rekhter. *RFC 1105: Border Gateway Protocol (BGP)*, June 1989.

[29] K. Lougheed and Y. Rekhter. *RFC 1163: Border Gateway Protocol (BGP)*, June 1990.

[30] K. Lougheed and Y. Rekhter. *RFC 1267: Border Gateway Protocol 3 (BGP)*, October 1991.

[31] Y. Rekhter and T. Li. *RFC 1654: A Border Gateway Protocol (BGP)*, July 1994.

[32] V. Fuller, T. Li, J. I. Yu, and K. Varadhan. *RFC 1519: Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy*, September 1993.

[33] *Intermediate System to Intermediate System Intra-Domain Routeing Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)*, February 1990.

[34] C. L. Hedricks. *An Introduction to IGRP.* Center for Computer and Information Services, Laboratory for Computer Science Research, Rutgers University, August 1991.

[35] D. Farinacci. *Introduction to Enhanced IGRP (EIGRP).* Cisco Systems, July 1993.

[36] E. Royer and C. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. In *IEEE Personal Communications*, volume 6(2), pages 46–55, April 1999.

[37] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance Vector (DTDV) for Mobile Computers. In *Proc. of the SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.

[38] S. Murthy and J. J. Garcia-Luna-Aveces. A Routing Protocol for Packet Radio Networks. In *Proc. ACM International Conference on Mobile Computing and Networking*, pages 86–95, November 1995.

[39] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. In *IEEE Singapore International Conference on Networks, SICON'97*, pages 197–211, Singapore, April 1997. IEEE.

[40] J. J. Garcia-Luna and M. Spohn. Source-Tree Routing in Wireless Networks. In *Proceedings of the 7th International Conference on Network Protocols, IEEE ICNP 99*, pages 273–282, Toronto, Canada, October 1999.

[41] R. Dube, C. D. Rais, K. Wang, and S. K. Tripathi. Signal Stability based adaptive routing (SSR alt SSA) for ad hoc mobile networks. In *IEEE Personal Communication*, February 1997.

[42] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, 353:153–181, 1996.

[43] C. Perkins, E. Royer, and S. Das. *RFC 3561: Ad hoc On-demand Distance Vector (AODV) Routing*, July 2003.

[44] C. Toh. A Novel Distributed Routing Protocol To Support Ad hoc Mobile Computing. In *Proc. IEEE 15th Annual International Phoenix Conference on Computers*

and Communications, IEEE IPCCC 1996, pages 480–486, Phoenix, AZ, USA, March 1996.

[45] Z. J. Haas, M. R. Pearlman, and P. Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks, July 2002. Internet Draft (work in progress).

[46] T. Chen and M. Gerla. Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks. In *Proc. IEEE ICC'98*, pages 171–175, Atlanta, GA, USA, June 1998.

[47] N. Nikaein and C. Bonnet. Hybrid Ad Hoc Routing Protocol - HARP. In *Proceedings of IST 2001: International Symposium on Telecommunications*, 2001.

[48] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm. In *The 18th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM '99*, pages 202–209, New York, NY, USA, March 1999.

[49] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *Proc. ACM/IEEE Mobicom*, pages 76–84, October 1998.

[50] D. Câmara and A. A. F. Loureiro. A Novel Routing Algorithm for Hoc Networks. *Baltzer Journal of Telecommunications Systems*, 18:1-3:85–100, 2001.

[51] I. Lu M. Joa-Ng. A Peer-to-Peer Zone-Based Two-Level Link State Routing for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas In Communication*, 17(8):1415–1425, August 1999.

[52] A. Lingdren and O. Schelén. Infrastructured ad hoc networks. In *Proceedings of the 2002 International Conference on Parallel Processing Workshops (International Workshop on Ad Hoc Networking (IWAHN 2002))*, pages 64–70, August 2002.

[53] L. Ji and M. S. Corson. A Lightweight Adaptive Multicast Algorithm. In *Proc. of the IEEE Global Telecommunications Conference (Globecom)*, pages 1036–1042, Sydney, Australia, November 1998.

[54] V. Devarapalli, A. A. Selcuk, and D. Sidhu. MZR: A Multicast Protocol for Mobile Ad Hoc Networks. In *Proc. of the IEEE International Conference on Communications (ICC)*, pages 886–891, Helsinki, Finland, June 2001.

[55] H. Xiao, W. K. G. Seah, A. Lo, and K. C. Chua. A Flexible Quality of Service Model for Mobile Ad Hoc Networks. In *Proceedings of IEEE Vehicular Technology Conference*, pages 445–449, Tokyo, Japan, May 2000.

[56] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. *RFC 2386: A Framework for QoS-based Routing in the Internet*, August 1998.

[57] S. Chen and K. Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, November 1998.

[58] D. Medhi. A Perspective on Network Restoration. *Handbook of Optimization in Telecommunications*, 2005.

[59] Q. Ma and P. Steenkiste. Quality of Service routing for traffic with performance guarantees. In *Proceedings of WQoS*, 1997.

[60] A. Shaikh, J. Rexford, and K. Shin. Evaluating the impact of stale link state on Quality of Service routing. *IEEE/ACM Transactions on Networking*, 9(2):162–176, 2001.

[61] S. Siachalou and L. Georgiadis. Efficient QoS routing. In *Proceedings of IEEE INFOCOM*, 2003.

[62] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proceedings of IEEE INFOCOM*, pages 2129–2133, 1995.

[63] Y. Cui and J. Wu. Clustering-based distributed precomputation for Quality of Service routing. In *Proceedings of Intl. Conference on Computational Science*, 2005.

[64] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson. Algorithms for computing QoS paths with restoration. In *Proceedings of IEEE INFOCOM*, 2003.

[65] Java Technology. `http://java.sun.com`.

[66] Applets. `http://java.sun.com/applets`.

[67] Creating a GUI with JFC/Swing. `http://java.sun.com/docs/books/tutorial/uiswing`.

[68] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, pages 251–262, Cambridge, Massachusetts, USA, 1999. ACM Press.

[69] A. Medina, I. Matta, and J. Byers. On the Origin of Power Laws in Internet Topologies. *SIGCOMM Computer Communication Review*, 30(2):18–28, 2000.

[70] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The Origin of Power-Laws in Internet Topologies Revisited. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2002)*, 2002.

[71] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network Topology Generators: Degree-Based vs. Structural. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'2002)*, pages 147–159, 2002.

[72] T. Bu and D. F. Towsley. On Distinguishing between Internet Power Law Topology Generators. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2002)*, 2002.