MAVERSON EDUARDO SCHULZE ROSA

UM ALGORITMO HEURÍSTICO PARA ROTEAMENTO ROBUSTO E EFICIENTE BASEADO EM AVALIAÇÃO DE FLUXO MÁXIMO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2008

Agradecimentos

Agradeço principalmente a Deus por me capacitar e por estar comigo durante esta caminhada. Agradeço também ao Prof. Elias Procópio Duarte Jr. por todo o esforço e atenção empregados na orientação deste mestrado. Não posso deixar de agradecer aos meus pais, por terem me apoiado incondicionalmente em todos os momentos. Agradeço também à minha esposa, Daniely, pela paciência que teve durante os meus longos períodos de dedicação ao mestrado. Agradeço aos amigos, em especial ao João Gustavo, à Ana Flávia e ao Juliano Picussa, pela prontidão em ajudar e pela motivação nas horas de desânimo. Agradeço à Andrea Weber pela atenção e pelo modo como compartilhou alguns dos conhecimentos obtidos em seu doutorado. Também quero agradecer à UFPR e à CAPES pelo apoio institucional e financeiro que deram a este trabalho.

SUMÁRIO

Lista de Figuras

Glossário de Siglas

AS	*Autonomous System*
BGP	*Border Gateway Protocol*
CIDR	*Classless Inter-Domain Routing*
EGP	*Exterior Gateway Protocol*
EGP's	*Exterior Gateway Protocols*
IAB	*Internet Architecture Board*
IETF	*Internet Engineering Task Force*
IGP's	*Interior Gateway Protocols*
IGRP	*Interior Gateway Routing Protocol*
IP	*Internet Protocol*
IPv6	*Internet Protocol versão 6*
IS-IS	*Intermediate System to Intermediate System*
MPLS	*Multi-Protocol Label Switching*
OSI	*Open Systems Interconnection*
OSPF	*Open Shortest Path First*
QoS	*Quality of Service*
RFC	*Request For Comments*
RIP	*Routing Information Protocol*
RIP-2	*Routing Information Protocol versão 2*
RIP-2 MIB	*RIP-2 Management Information Base*
RIPng	*RIP next generation*
SPF	*Shortest Path First*
TCP	*Transmission Control Protocol*
ToS	*Type of Service*
UDP	*User Datagram Protocol*

Resumo

Os protocolos de roteamento são baseados em tabelas de roteamento e, à medida em que ocorrem alterações na topologia da rede, essas tabelas permanecem desatualizadas durante um certo período de tempo (latência de convergência), até que as informações sejam atualizadas. No BGP (*Border Gateway Protocol*), o protocolo de roteamento externo utilizado na Internet, esse período é de aproximadamente três minutos, sendo que já foram observados picos de até quinze minutos. Durante esse período, pacotes de dados podem ser perdidos e conexões podem ser interrompidas. Recentemente, foi proposta uma estratégia de roteamento dinâmico tolerante a falhas que possibilita o correto funcionamento do roteamento durante esse período. Essa estratégia baseia-se na utilização de um critério de fluxo máximo, utilizado em conjunto com outros critérios e um mecanismo de *backtracking*, para avaliar os caminhos possíveis em uma rede. O presente trabalho propõe um algoritmo heurístico que possibilita a utilização de tabelas de roteamento que correlacionam um endereço de destino à múltiplos endereços de saída, ordenados de acordo com estimativas da robustez dos caminhos. Essas tabelas propiciam o *backtracking* de mensagens durante o processo de encaminhamento, em contraste com a estratégia anterior, que realiza um cálculo completo das estimativas a cada mensagem encaminhada. São também apresentados uma prova formal da correção do algoritmo proposto e resultados de simulações comparando-o com outras alternativas. Esses resultados comprovam que, com uma alteração no formato das tabelas de roteamento e um pequeno aumento no comprimento médio dos caminhos percorridos, é possível efetuar o roteamento robusto de mensagens de forma eficiente.

Abstract

Several routing protocols are based on routing tables that must be updated as the network topology changes. From the instant of time the change occurs and the tables are updated (convergence latency), packets can be dropped and connections broken. In the BGP (Border Gateway Protocol), the Internet exterior routing protocol, this period is approximately three minutes, and peaks up to fifteen minutes have been observed. Recently a fault-tolerant routing algorithm that improves the correctness of routing during this period was proposed. This algorithm uses a backtracking mechanism combined with maximum flow evaluation to select network paths. Nevertheless the algorithm as it is proposed cannot be deployed to real networks. The present work is based on that algorithm, but proposes a new heuristic algorithm that allows routing tables to be used. These tables correlate multiple output addresses, ordered according estimates of robustness and enable backtracking during the forwarding process. This contrasts with the previous algorithm, which performs a full evaluation for each forwarded message. A correctness proof and experimental results obtained with simulation are also presented, comparing the algorithm to Dijkstra's shortest-path first algorithm and to the previous algorithm. These results show that robust routing is more efficient with small changes in the format of currently used routing tables and with a small increase of the average length of shortest paths.

Capítulo 1

Introdução

A Internet é uma coleção de redes interconectadas através de roteadores. Estes, por sua vez, estão organizados de forma hierárquica, em que alguns roteadores são utilizados apenas para trocar dados entre grupos de redes controlados pela mesma autoridade administrativa, enquanto outros roteadores fazem também a comunicação entre as autoridades administrativas. A entidade que controla e administra um grupo de redes e roteadores é chamada de Sistema Autônomo (AS - *Autonomous System*) [10].

O processo de seleção de caminhos para o envio de dados entre pontos distintos dentro de um AS, ou entre diversos AS's, é conhecido como roteamento. Os dados podem ser roteados de sua fonte até o destino através de uma série de roteadores e, eventualmente, entre múltiplos AS's. O principal modelo de roteamento utilizado é o do *hop-by-hop* [10], onde cada roteador que recebe um pacote de dados, verifica o endereço de destino desse pacote, determina o endereço do próximo roteador em direção ao destino e entrega o pacote para esse roteador. Esse processo se repete até a entrega do pacote ao seu destinatário.

Para determinar o endereço do próximo roteador em direção ao destino final, um roteador efetua uma consulta à sua tabela de roteamento. As tabelas de roteamento associam o endereço de cada destino ao endereço de próximo *hop*. Essas tabelas são montadas de acordo com as informações da topologia da rede conhecidas em cada roteador.

Os protocolos de roteamento são responsáveis por manter essas informações e gerar as

tabelas de roteamento em conformidade com as informações conhecidas. Porém, à medida em que ocorrem alterações na topologia da rede, essas informações ficam defasadas e, conseqüentemente, as tabelas de roteamento permanecem desatualizadas por um certo período de tempo até que as informações sejam atualizadas. O período de tempo entre a ocorrência de uma alteração na topologia até a atualização completa de todas as tabelas de roteamento é conhecido como latência de convergência [32] do protocolo.

A inconsistência das tabelas de roteamento nos roteadores de uma rede durante esse período de tempo pode causar a perda de pacotes e a interrupção de conexões. No OSPF (*Open Shortest Path First*), um dos principais protocolos para o roteamento interno a um AS, a latência de convergência pode se prolongar por cerca de um minuto, como detalhado no capítulo 2. Já no BGP (*Border Gateway Protocol*), o protocolo de roteamento externo usado na Internet, essa latência é de aproximadamente três minutos, sendo que já foram observados picos de até quinze minutos [32].

Para resolver os problemas gerados pela alta latência de convergência dos protocolos de roteamento utilizados na Internet, foram propostas e implementadas diversas soluções. Por um lado, existem soluções que procuram aperfeiçoar os protocolos existentes, como a de [2], em que são propostas apenas algumas alterações nos *timers* do OSPF. Por outro lado, existem soluções como a de Wei Li [33], em que se utilizam novas abordagens para otimizar o roteamento na Internet. Outras estratégias têm sido propostas, e algumas são descritas na seção 2.6.

Dentre as novas abordagens para o roteamento robusto, destaca-se a seleção de rotas baseada em critérios de conectividade, que foi originalmente apresentada em [17]. Em [50], é proposto um algoritmo para roteamento dinâmico tolerante a falhas baseado no critério de conectividade de fluxo máximo. As principais características desse algoritmo são descritas a seguir.

O algoritmo de roteamento recebe como entrada um par de nós da rede, correspondentes à origem e ao destino de uma mensagem a ser enviada. Ele é dinâmico, ou seja, é executado em cada nó, iniciando pelo nó de origem e determinando o próximo nó da rota dentre os nós vizinhos. Quando a mensagem chega ao nó escolhido, esse executa o mesmo

procedimento para escolher o nó seguinte, até que a mensagem chegue ao nó de destino. A adoção dessa abordagem implica em um algoritmo com maior robustez, pois as alterações na topologia da rede são, comumente, refletidas mais rapidamente em roteadores mais próximos. Então, conforme uma mensagem está sendo roteada, cada roteador mais próximo ao destino final tem uma representação mais atualizada da topologia da rede naquele ponto, podendo assim efetuar a escolha de uma rota mais adequada.

Esse algoritmo é tolerante a falhas, pois, baseia-se em critérios de conectividade para selecionar rotas robustas que possibilitem, na ocorrência de falhas, a utilização de outro caminho, um desvio, que parte do ponto em que a falha é conhecida em direção ao destino final da mensagem. Para a seleção de rotas, o comprimento dos caminhos é utilizado como critério secundário. O principal critério utilizado para a seleção de caminhos é o corte mínimo, ou fluxo máximo, entre o nó avaliado e o destino [22]. Esse critério é utilizado para avaliar a redundância dos caminhos, visto que, quanto maior o fluxo máximo, maior a quantidade de caminhos disjuntos e, conseqüentemente, maior é o número de desvios que podem ser utilizados em caso de falha. Desse modo, a seleção de uma aresta é feita baseando-se em um cálculo de compromisso (*trade-off*), descrito detalhadamente na seção 3.2. Esse cálculo baseia-se na avaliação da redundância e do comprimento do menor caminho que cada aresta propicia. Após a avaliação dos valores para cada aresta adjacente, é escolhida a aresta que possui o melhor resultado.

Esse algoritmo calcula o fluxo máximo efetivo entre cada nó avaliado para roteamento e o destino de cada mensagem. Para calcular o fluxo máximo efetivo são desconsiderados os caminhos disjuntos que utilizem nós já visitados por cada mensagem. Dessa forma, o processo de roteamento é dependente do contexto de cada mensagem, mais especificamente, do conjunto de nós visitados por cada mensagem a cada nó. Essa dependência implica em um custo de $O(E^3)$, com E sendo o número de arestas da rede, para o roteamento de cada mensagem. Uma complexidade $O(E^3)$ por mensagem roteada não propicia uma implementação prática do algoritmo de roteamento. Assim sendo, a proposta apresentada neste trabalho visa tornar possível a escolha do próximo nó de uma rota, considerando os critérios de avaliação do algoritmo anterior, apenas consultando a mensagem a ser roteada

e a tabela de roteamento do nó que deseja rotear essa mensagem.

Pata tanto, o algoritmo proposto utiliza-se de uma heurística para mensurar o critério de fluxo máximo, o que efetivamente permite a construção de uma tabela de roteamento em cada nó da rede. Neste sentido, a complexidade do roteamento é transferida para a construção destas tabelas.

Como no algoritmo Schroeder-Duarte, apresentado em [50], o algoritmo proposto baseia-se em representações da topologia mantidas individualmente por cada nó pertencente à rede. Esse algoritmo também escolhe, em cada nó percorrido, apenas a próxima aresta do caminho até o destino. Por isso, não é necessário que a representação da topologia, e conseqüentemente a tabela de roteamento, mantida pelo nó que executa o algoritmo reflita as últimas alterações ocorridas na rede.

O procedimento de manutenção da topologia da rede em cada nó, apresentado em [50], foi modificado de forma a englobar o algoritmo responsável pela geração da tabela de roteamento. Esse algoritmo é executado por um nó s sempre que ocorrer uma atualização na sua representação da topologia, e gera como resultado uma tabela de roteamento contendo todos os destinos possíveis no estilo da Internet [10]. Porém, para cada entrada, ou destino, da tabela de roteamento é mantida uma lista de arestas candidatas ao roteamento que armazena somente as arestas adjacentes, ao nó s, que possuam um caminho até o destino. Essa lista é ordenada conforme a função de avaliação apresentada na seção 3.2.

Com a tabela de roteamento montada em cada nó, é possível efetuar a escolha da aresta que deve ser utilizada para o roteamento de uma mensagem apenas consultando a tabela gerada previamente e a mensagem a ser roteada. Desse modo, para rotear uma mensagem, cada nó consulta a sua tabela, obtém a aresta que leva ao próximo nó da rota e encaminha a mensagem para esse nó. Esse procedimento é executado até que o destino final seja alcançado.

Uma especificação detalhada dos procedimentos de manutenção de topologia, de geração de tabelas de roteamento, de envio e recebimento de mensagens de atualização de topologia e de encaminhamento de mensagens é apresentada no decorrer deste trabalho. Além

dessa especificação é apresentada uma prova formal da correção do algoritmo proposto.

O funcionamento desse algoritmo é comparado, através de simulações, com o algoritmo de roteamento baseado no algoritmo de Dijkstra [15] e com o algoritmo Schroeder-Duarte [50]. Nestas comparações são considerados os seguintes aspectos: comprimento médio de caminhos percorridos por mensagens entregues; comprimento médio dos caminhos percorridos por mensagens entregues que efetuaram *backtracking*; número de mensagens descartadas; e a quantidade efetuada de cálculos de avaliação de aresta. Os resultados dessas comparações comprovam que, com uma alteração no formato das tabelas de roteamento e um pequeno aumento no comprimento médio dos caminhos percorridos, é possível efetuar o roteamento robusto de mensagens de forma eficiente.

Este trabalho está organizado como segue. O capítulo 2 descreve como é efetuada a seleção de caminhos para a entrega de pacotes entre os diversos componentes que constituem uma rede. Também são descritos nesse capítulo alguns dos trabalhos que visam aperfeiçoar os métodos de roteamento existentes, bem como trabalhos que introduzem novas abordagens ao problema de roteamento. No capítulo 3 é apresentado o algoritmo de roteamento baseado em critérios de conectividade, Schroeder-Duarte, [50] no qual é baseada a solução proposta. O capítulo 4 apresenta o algoritmo heurístico que permite a elaboração de um protocolo prático para o roteamento tolerante a falhas baseado nos critérios definidos no capítulo 3. No capítulo 5 são apresentados os resultados obtidos através da simulação desse algoritmo comparando-o ao algoritmo de Dijkstra e ao algoritmo Schroeder-Duarte. Por fim, o capítulo 6 apresenta a conclusão e os próximos passos deste trabalho.

Capítulo 2

Roteamento na Internet

Este capítulo apresenta a arquitetura de roteamento da Internet, descrevendo como é efetuada a seleção de caminhos para a entrega de pacotes entre os diversos componentes que constituem essa rede. Neste capítulo também são apresentados trabalhos recentes relacionados à proposta desta dissertação.

O capítulo está organizado da seguinte maneira. A seção 2.1 define os conceitos básicos de roteamento e teoria dos grafos utilizados no trabalho. A seção 2.2 introduz a arquitetura de roteamento da Internet. A seção 2.4 apresenta os principais protocolos e algoritmos utilizados para o roteamento interno a um AS. A seção 2.5 explica o funcionamento do principal protocolo utilizado no roteamento entre AS's. Por fim, a seção 2.6 apresenta os principais trabalhos relacionados.

2.1 Definições - Conceitos Básicos

Esta seção apresenta alguns conceitos básicos e definições de teoria dos grafos [25, 14] e roteamento. Esses conceitos serão utilizados no decorrer deste trabalho.

Definição 2.1 (Grafo): Um grafo $G = (V, E)$ é definido pelo par de conjuntos V e E, onde V é um conjunto não vazio, cujos elementos são chamados vértices, e E é um conjunto de arestas. Uma aresta é um par não ordenado $\{u,v\}$, em que u e v são elementos de V.

Em um grafo direcionado, também chamado digrafo, uma aresta é um par ordenado

(u,v). Diz-se que cada aresta (u,v) possui uma única direção de u para v. Também a aresta (u,v) é dita divergente de u e convergente a v. Os números de arestas divergentes e convergentes de um vértice são chamados de grau de saída e grau de entrada, respectivamente.

Para o problema de roteamento, um grafo representa a rede de computadores como um todo, onde os vértices, nós, representam as máquinas (*hosts* e roteadores) e as arestas representam os enlaces da rede.

Definição 2.2 (Adjacência): Seja $G = (V, E)$ um grafo; sejam $u, v \in V$ nós do grafo G; e seja $e \in E$ uma aresta nesse 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 2.3 (Caminho): Seja $G = (V, E)$ um grafo; sejam s e $t \in V$ nós do grafo G. Um caminho entre a origem s e o destino t é uma seqüência de nós distintos $p = (v_0, v_1, ..., v_{n-1}, v_n)$ 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 $v \in \{v_0, v_1, ..., v_n\}$. Dizemos que $e \in E$ pertence a p se $\exists i, 0 \leq i < n, e = (v_i, v_{i+1})$.

A palavra *rota* é utilizada, neste trabalho, como sinônimo para caminho. Esse termo é comumente utilizado em trabalhos relacionados a redes de computadores para especificar o caminho usado para enviar uma mensagem de nó de origem a um nó de destino. Para problemas como o de *fluxo* em redes é comum chamar de nó produtor o nó origem s, e chamar de nó consumidor, ou depósito, o nó destino t.

Definição 2.4 (Ciclo): Seja $G = (V, E)$ um grafo; seja $p = (v_0, v_1, ..., v_{k-1})$ um caminho não trivial em G. Se existir a aresta (v_{k-1}, v_0) e $k \geq 3$, então o subgrafo C contendo os vértices e arestas do caminho p mais a aresta (v_{k-1}, v_0) é chamado um *ciclo*.

Como freqüentemente denotamos um ciclo pela sua seqüência, cíclica, de vértices; o ciclo C, descrito acima, pode ser expresso como $v_0, v_1, ..., v_{k-1}, v_0$.

Definição 2.5 (Rede): Uma rede é um grafo $G = (V, E)$, sendo V o conjunto de

vértices e E o conjunto de arestas de G, onde cada uma das arestas $e \in E$ tem uma capacidade $c(e) \geq 0$, onde $c : E \rightarrow \Re$. Se $e \notin E$ então assume-se que $c(e) = 0$. Além disso, em uma rede são definidos dois vértices distintos chamados origem s e destino t.

Definição 2.6 (Roteamento): Seja $G = (V, E)$ a representação de uma rede em um grafo; sejam s e $t \in V$ nós do grafo G, correspondentes ao nó origem e ao nó destino, respectivamente. Se s precisa enviar uma mensagem para t, é chamado de roteamento o processo de escolha de um caminho entre s e t.

2.2 Arquitetura de Roteamento na Internet

A Internet é uma coleção de redes interconectadas, e os pontos de ligação dessas redes são os roteadores. Esses, por sua vez, estão organizados de forma hierárquica, onde alguns roteadores são utilizados apenas para trocar dados entre grupos de redes controlados pela mesma autoridade administrativa; enquanto outros roteadores fazem também a comunicação entre as autoridades administrativas. A entidade que controla e administra um grupo de redes e roteadores é chamada de Sistema Autônomo (AS) [10].

Os AS's podem ser classificados em três tipos distintos: *Stub AS's*, *Transit AS's* e *Multihomed AS's*. Um *Stub AS* possui uma única conexão com outro AS. Todos os pacotes de dados enviados, ou recebidos, para fora desse AS devem ser enviados por essa conexão. Um *Transit AS* possui múltiplas conexões com um ou mais AS's, o que permite a pacotes de dados, que não possuam como destino um ponto interior ao AS, trafegar por esse AS. Um *Multihomed AS* também possui múltiplas conexões com um ou mais AS's. Porém, não é permitido que pacotes de dados recebidos por uma dessas conexões sejam encaminhados para fora do AS novamente. Ou seja, esse tipo de AS não provê um serviço de trânsito para outros AS's.

O processo de seleção de caminhos para o envio de dados entre pontos distintos dentro de um AS, ou entre diversos AS's, é conhecido como roteamento. Os dados podem ser roteados de sua fonte até o destino através de uma série de roteadores e, eventualmente, entre múltiplos AS's. O modelo de roteamento comumente utilizado é o do *hop-by-hop*, onde cada roteador que recebe um pacote de dados, verifica o endereço de destino desse

pacote, calcula endereço do próximo *hop* em direção ao destino e entrega o pacote para esse *hop*. Esse processo se repete até a entrega do pacote ao seu destinatário. No entanto, para isso, são necessários dois elementos: as tabelas de roteamento e um protocolo de roteamento, descritos a seguir.

As tabelas de roteamento correlacionam cada endereço de destino com o endereço do próximo *hop*. Os protocolos de roteamento habilitam os roteadores a construírem suas tabelas de roteamento. Esses protocolos determinam a forma como a tabela é montada e quais as informações que a compõem. Dentre os principais tipos de protocolos podemos citar dois tipos: os que utilizam algoritmos baseados em vetores de distância (*Distance-Vector Routing Protocols*) e os que utilizam algoritmos baseados no estado de enlace (*Link-State Routing Protocols*) [26].

Além de ser classificado quanto ao tipo de algoritmo utilizado para o roteamento, um protocolo também pode ser classificado quanto à sua abrangência. Os protocolos utilizados para rotear dados internamente a um AS são ditos protocolos de roteamento internos (IGP's - *Interior Gateway Protocols*). Os IGP's habilitam diferentes redes dentro de um AS a transmitirem informações entre si. Eles também habilitam o encaminhamento de pacotes de dados através do AS, caso seja um *transit AS*. Os protocolos utilizados para rotear dados entre AS's distintos são ditos protocolos de roteamento externo (EGP's - *Exterior Gateway Protocols*). Eles habilitam roteadores dentro de um AS a escolher o melhor ponto de saída para o AS de destino dos pacotes que estão sendo roteados.

O EGP e os IGP's executando em cada AS precisam cooperar para rotear pacotes através da Internet. O EGP determina os AS's que os pacotes devem cruzar para alcançar um destino, e os IGP's determinam o caminho dentro de cada AS que os pacotes devem seguir para ir do ponto de origem até o destino final.

2.3 Algoritmos de Roteamento

Para determinar o caminho entre nós de uma rede, os *protocolos* de roteamento utilizam diversos *algoritmos* de roteamento. Ou seja, cada protocolo de roteamento implementa um algoritmo de roteamento. Esses algoritmos de roteamento podem ser diferenciados de

acordo com suas características, sendo que essas características que determinam o modo de operação dos protocolos que os implementam.

Entre as características desejáveis aos algoritmos de roteamento [16], destacam-se: otimização, simplicidade, baixo *overhead*, robustez, convergência rápida e flexibilidade. A otimização refere-se à capacidade do algoritmo em selecionar as melhores rotas possíveis, o que depende das métricas e pesos utilizados nos cálculos de rotas. Simplicidade e baixo *overhead* são particularmente importantes quando se deseja executar o algoritmo em ambientes heterogêneos, como a Internet, em que é necessário utilizar o mínimo possível de recursos. A robustez está relacionada ao correto funcionamento dos algoritmos em situações de adversidade, como falhas em dispositivos, congestionamento da rede, divergência de informações nos nós da rede, entre outras. A convergência rápida refere-se à capacidade dos roteadores acordarem rapidamente sobre as melhores rotas possíveis, evitando, por exemplo, que aconteçam *loops* e perda de pacotes. A flexibilidade expressa a capacidade de adaptação do algoritmos às diversas situações das redes.

Os algoritmos de roteamento podem ser classificados de acordo com uma série de diferenciadores, dentre os quais podemos citar: estáticos/dinâmicos, caminhos simples/múltiplos (*single-path/multipath*), hierárquico/não hierárquico, estado de enlace/vetor-distância, entre outros. Essas classificações são apresentadas a seguir.

Os algoritmos estáticos são constituídos basicamente em mapeamentos da rede definidos previamente ao início do roteamento. Esses mapeamentos não mudam, a não ser que alguém os altere manualmente. Como o roteamento, nesse caso, não reage à mudanças na rede, esse tipo de algoritmo é considerado inapropriado para as redes atuais, que apresentam alterações constantes em suas topologias. Dessa forma, atualmente, são utilizados algoritmos dinâmicos de roteamento, os quais redefinem os caminhos utilizados para o roteamento de acordo com a detecção de mudanças na topologia, alterações no tráfego ou baseado em outras métricas possíveis [16].

Uma classificação quanto aos tipos de caminhos utilizados pelos algoritmos de roteamento também é utilizada. Os algoritmos podem tanto utilizar caminhos simples, ou únicos, para realizar o roteamento das mensagens entre uma origem e um destino (*single-*

path), quanto efetuar uma distribuição da carga entre vários caminhos possíveis (*multi-path*).

Também é possível efetuar a classificação quanto à utilização de uma hierarquia diferenciada entre os roteadores da rede. Nos algoritmos de roteamento hierárquicos, alguns roteadores se agrupam em um conjunto especial de roteadores interligados (*backbone*). Pacotes enviados por roteadores que não pertencem ao *backbone* são encaminhados aos roteadores do *backbone*. Ao chegar em um desses roteadores, esses pacotes são roteados através do *backbone* até chegarem na área em que o destino final está localizado. Nesse ponto, eles são roteados através dos roteadores não pertencentes ao *backbone* até o destino final.

Os algoritmos da categoria vetor-distância (*Distance-Vector*) são também conhecidos como algoritmos de Bellman-Ford, o primeiro algoritmo dessa categoria foi originalmente apresentado em [3]. Nesses algoritmos cada roteador não processa informações sobre a topologia completa da rede, ele apenas anuncia suas distâncias para roteadores adjacentes e recebe, de forma similar, anúncios de outros roteadores. Usando esses anúncios, cada roteador preenche sua tabela de roteamento. Em cada ciclo de anúncios, um roteador anuncia todas as informações atualizadas de sua tabela de roteamento. Esse processo continua até que as tabelas de roteamento de cada roteador convirjam para valores estáveis.

Esse conjunto de algoritmos apresenta como desvantagem, em relação aos algoritmos de estado de enlace, um maior tempo de convergência. Entretanto, eles são utilizados na Internet por apresentarem uma grande vantagem: não necessitam da topologia completa da rede. Exemplos de protocolos que utilizam esse tipo de algoritmo são: o RIP (*Routing Information Protocol*) [27] e o BGP (*Border Gateway Protocol*) [47].

Já na categoria de estado de enlace (*Link-State*), também conhecida como roteamento pelo caminho mais curto (SPF - *Shortest Path First*) ou Dijkstra [15], cada nó da rede processa informações sobre a topologia completa da rede. Então, cada nó calcula o melhor próximo *hop* para cada possível destino da rede, utilizando para isso as informações que possui sobre a topologia. A coleção dos melhores próximos *hops* forma a tabela de roteamento para o nó.

Em contraposição aos algoritmos vetor-distância, para a manutenção das tabelas de roteamento, os algoritmos de estado de enlace transmitem unicamente as informações necessárias para construir as representações da topologia da rede em cada nó. Exemplos de protocolos que utilizam dessa categoria de algoritmos são: o OSPF (*Open Shortest Path First*) [42] e o IS-IS (*Intermediate System to Intermediate System*) [44].

A seguir é descrito em mais detalhes o funcionamento dos dois principais algoritmos de roteamento em uso na Internet, o Bellman-Ford e o Dijkstra.

2.3.1 Funcionamento do Algoritmo de Vetor-Distância (Bellman-Ford)

A família de algoritmos vetor-distância é também conhecida como de Bellman-Ford, pois baseia-se no algoritmo descrito por R. E. Bellman [3], e a primeira descrição do algoritmo distribuído é atribuída a Ford e Fulkerson [22]. A versão centralizada do algoritmo de vetor-distância é apresenta a seguir. Logo após é ilustrado o funcionamento da versão distribuída através de um exemplo simples.

O algoritmo de Bellman-Ford calcula os caminhos mínimos a partir de um único vértice em um digrafo ponderado (*weighted digraph*) [25]. A figura 2.1 apresenta a versão centralizada desse algoritmo utilizada para calcular as tabelas de roteamento de todos os nós de uma rede, conforme descrito em [30].

Inicialmente, a matriz de distâncias é marcada com zero nas posições que representam a distância para um mesmo nó, e com infinito na posições que representam a distância entre nós diferentes. E então, para cada aresta, é avaliada a distância para todos os destinos considerando o seu peso. Caso a distância, considerando o peso da aresta avaliada seja menor que a distância previamente marcada, a distância é atualizada. Toda vez que uma distância for atualizada, é necessário reavaliar as distâncias até que não ocorra mais nenhuma atualização.

Quando é atingido o ponto em que não são mais efetuadas atualizações nas distâncias, diz-se que o algoritmo convergiu. Esse algoritmo converge, no máximo, em N iterações, com N sendo o número de nós da rede. E, a complexidade de cada iteração é $O(N * M)$,

```
CentralizedBellman-Ford( G = (V,E) )
```

1. Let $N \leftarrow$ the number of nodes in V
2. Let $M \leftarrow$ the number of edges in E
3. Let $A \leftarrow$ a table with size M, where $A[a].m$, $A[a].s$ and $A[a].d$ are the metric, source, and destination of edge a, respectively
4. Let $D \leftarrow$ a matrix with size $N \times N$, where $D[i, j]$ is the distance from node i to node j
5. Let $H \leftarrow$ a matrix with size $N \times N$, where $H[i, j]$ represents the edge used by i to route packages destined to j
6. For each position $D[i, j]$
7. If $i = j$
8. $D[i, j] = 0$
9. Else
10. $D[i, j] = \infty$
11. For each edge $a \in E$
12. For each destination $k \in V$
13. Let $i \leftarrow A[a].s$
14. Let $j \leftarrow A[a].d$
15. Let $d \leftarrow A[a].m + D[j, k]$
16. If $d < D[i, k]$
17. $D[i, k] \leftarrow d$
18. $H[i, k] \leftarrow a$
19. If any distance $D[i, k]$ has changed
20. Repeat from line 11

Figura 2.1: Versão centralizada do algoritmo Bellman-Ford.

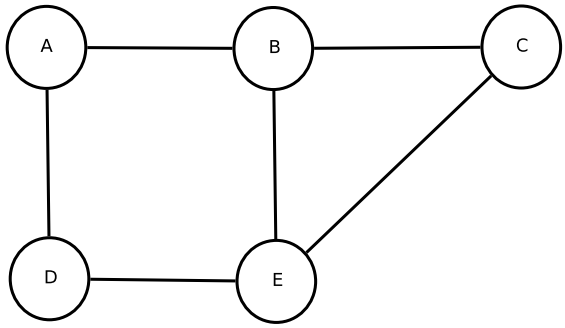

Figura 2.2: Exemplo de rede para demonstração do algoritmo Bellman-Ford.

com M sendo o número de arestas da rede. Portanto, a complexidade do algoritmo, no pior caso, é $O(M * N^2)$.

Porém, em sua versão distribuída, cada nó é responsável apenas por calcular a sua tabela de roteamento. Ou seja, cada nó i efetua o processamento considerando suas arestas de saída, as colunas $D[i, *]$ da matriz de distâncias e as colunas $H[i, *]$ da matriz de roteamento.

Para demonstrar o funcionamento desse algoritmo, é utilizada a rede apresentada em [30] e ilustrada pela figura 2.2. As arestas, para simplificar, são consideradas como sendo *links* simétricos, bidirecionais ($(A, B) = (B, A)$), e possuem como peso o valor 1. Os nós possuem a capacidade de encaminhar pacotes e não são diferenciados entre *hosts* e roteadores, eles tem como objetivo montar tabelas de roteamento que indiquem como alcançar os demais nós da rede.

É considerada a situação inicial de *cold start*, ou seja, os nós da rede são ativados simultaneamente. Cada nó, quando ativado possui apenas o conhecimento de seu próprio endereço na rede (representados pelas letras A, B, C, D e E) e dos *links* a que estão conectados. Isto é, possuem apenas o conhecimento de suas condições locais (*local knowledge*) e ignoram a topologia da rede como um todo.

De A para	*Link*	Custo
A	local	0

De B para	*Link*	Custo
B	local	0

De C para	*Link*	Custo
C	local	0

De D para	*Link*	Custo
D	local	0

De E para	*Link*	Custo
E	local	0

Figura 2.3: As tabelas de roteamento no momento inicial.

Nesse ponto, as tabelas de roteamento em cada nó possuem apenas uma única entrada apontando para si mesmo. A figura 2.3 apresenta o estado inicial das tabelas de roteamento em cada nó. As informações contidas em suas tabelas de roteamento são encaminhadas aos nós vizinhos através dos *links* locais. Para encaminhar as informações de roteamento é utilizada uma estrutura, denominada vetor-distância (*distance vector*), que lista os nós alcançáveis e as distâncias (custos) para alcançar esses nós, de acordo com a tabela de roteamento.

Considerando o nó A, por sua tabela possuir apenas uma entrada na situação inicial, o vetor-distância gerado possui apenas um componente: $A = 0$. Após gerar esse vetor-distância, o nó A o envia aos nós B e D através das arestas (A, B) e (A, D) respectivamente.

Na ocasião de recebimento de uma mensagem contendo o vetor-distância de um nó, é feita a atualização de todas as distâncias contidas no vetor. Essa atualização é feita adicionando o custo do *link* local, pelo qual a mensagem foi recebida, a todas as distâncias contidas no vetor. Após essa atualização, é efetuada uma comparação das distâncias atualizadas do vetor com os custos indicados na tabela de roteamento do nó receptor. Caso a distância contida no vetor seja menor que o custo indicado na tabela, a entrada correspondente ao nó comparado é alterada de acordo com as informações contidas no vetor. E, se um destino contido no vetor de distâncias não estiver representado por alguma entrada da tabela de roteamento, é adicionada uma nova entrada correspondendo a esse destino.

Dessa forma, ao receber o vetor do nó A, o nó D atualiza-o considerando o custo do *link* pelo qual foi recebido, ou seja o vetor resultado corresponderá a: $A = 1$, pois os custos dos *links* são considerados como 1. Então, como a tabela de roteamento do nó D não possui uma entrada para o nó A, é adicionada uma nova entrada conforme o vetor obtido. E, ao ajustar a sua tabela de roteamento o nó D prepara para o seu próprio vetor-distância e envia-o pelos seus *links* locais, ou seja: (D, A) e (D, E). O vetor-distância enviado pelo nó D possui as seguintes entradas: $D = 0, A = 1$.

De maneira semelhante, durante esse período, o nó B recebe o vetor de A e efetua o

mesmo processo enviando seu vetor-distância, $B = 0, A = 1$, pelos *links* (B, A), (B, E) e (B, C). Assim sendo, as mensagens enviadas por B são recebidas pelos nós A, E e C. Enquanto as mensagens enviadas pelo nó D são recebidas pelos nós A e E.

Supondo que a mensagem enviada por D chegue primeiro ao nó A. Esse irá, então, atualizar o vetor-distância enviado por D para: $D = 1, A = 2$. O passo seguinte é comparar o vetor resultante à tabela e, como não é encontrada nenhuma entrada para o nó D, é inserida uma nova entrada para esse nó. Ao comparar a distância $A = 2$ do vetor, o nó A verifica que essa distância é maior que a distância $A = 0$ indicada pela tabela e, nesse caso, a entrada existente na tabela não é alterada. Da mesma forma, ao receber a mensagem do nó B, o nó A adiciona apenas mais uma entrada, $B = 1$, à sua tabela.

De forma semelhante, o nó E recebe duas mensagens seguidas de atualização. Considerando, que o vetor recebido do nó B chegue primeiro, o nó E insere duas novas entradas em sua tabela, $B = 1, A = 2$. E, ao receber o vetor de D, constata que a distância $A = 2$ recebida pelo *link* (D, E) é exatamente igual à distância para o nó A através do *link* (B, E) e, portanto, apenas será adicionada uma entrada correspondente ao nó D em sua tabela.

O nó C recebe o vetor-distância $B = 0, A = 1$ através do *link* (B, C), e atualiza sua tabela adicionando entradas para os nós A e B. A figura 2.4 apresenta as tabelas de roteamento de todos os nós após essa primeira rodada de atualizações.

De A para	*Link*	Custo
A	local	0
B	(A, B)	1
D	(A, D)	1

De B para	*Link*	Custo
B	local	0
A	(B, A)	1

De C para	*Link*	Custo
C	local	0
A	(C, B)	2
B	(C, B)	1

De D para	*Link*	Custo
D	local	0
A	(D, A)	1

De E para	*Link*	Custo
E	local	0
A	(E, B)	2
B	(E, B)	1
D	(E, D)	1

Figura 2.4: As tabelas de roteamento após a primeira rodada de atualizações.

Nesse ponto, os nós A, C e E estão prontos para enviar seus vetores-distância atualizados nos seus *links* locais. O nó A envia o vetor $A = 0, B = 1, D = 1$ nos *links* (A, B)

e (A, D). O nó C envia o vetor $C = 0, B = 1, A = 2$ nos *links* (C, B) e (C, E). O nó E envia o vetor $E = 0, B = 1, A = 2, D = 1$ nos *links* (E, B), (E, C) e (E, D).

Essas mensagens irão causar mudanças nas tabelas de roteamento nos nós B, D e E, que, por sua vez, irão propagar seus novos vetores. O nó B, agora, envia o vetor $B = 0, A = 1, D = 2, c = 1, E = 1$ nos *links* (B, A), (B, E) e (B, C). O nó D envia o vetor $D = 0, A = 1, B = 2, E = 1$ nos *links* (D, A) e (D, E). O nó E envia o vetor $E = 0, B = 1, A = 2, D = 1, C = 1$ nos *links* (E, B), (E, C) e (E, D).

De A para	*Link*	Custo
A	local	0
B	(A, B)	1
C	(A, B)	2
D	(A, D)	1
E	(A, B)	2

De B para	*Link*	Custo
B	local	0
A	(B, A)	1
C	(B, C)	1
D	(B, A)	2
E	(B, E)	1

De C para	*Link*	Custo
C	local	0
A	(C, B)	2
B	(C, B)	1
D	(C, E)	2
E	(C, E)	1

De D para	*Link*	Custo
D	local	0
A	(D, A)	1
B	(D, A)	2
C	(D, E)	2
E	(D, E)	1

De E para	*Link*	Custo
E	local	0
A	(E, B)	2
B	(E, B)	1
C	(E, D)	1
D	(E, C)	1

Figura 2.5: As tabelas de roteamento após a primeira rodada de atualizações.

Essas mensagens geram alterações nas tabelas de roteamento dos nós A, C e D e, esses propagam seus novos vetores-distância. Porém, dessa vez, as mensagens enviadas não geram nenhuma atualização nas tabelas dos demais nós. E, nesse ponto diz-se que o algoritmo convergiu, ou seja, as tabelas de roteamento dos nós representam adequadamente a topologia da rede. A figura 2.5 mostra as tabelas de roteamento dos nós da rede nesse momento.

Suponha que, nesse momento, o *link* (A, B) tenha falhado e a figura 2.6 representa a nova topologia da rede. Considere que os nós A e B tomem conhecimento da falha. Nesse caso, eles devem atualizar suas tabelas de roteamento para representar a nova topologia. Isso é feito através da inserção de um custo infinito em todas as entradas da tabela que utilizem o *link* falho. A figura 2.7 representa o estado das tabelas de roteamento dos nós A e B após a detecção da falha.

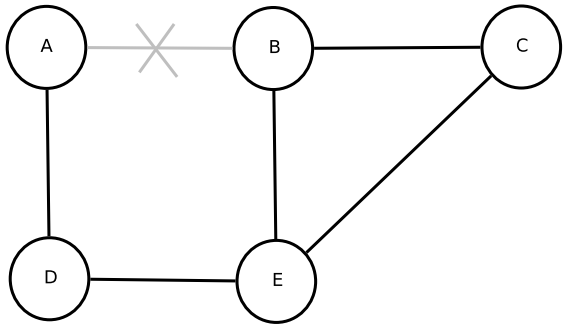

Figura 2.6: Exemplo de falha em um *link* da rede.

De A para	*Link*	Custo
A	local	0
B	(A, B)	∞
C	(A, B)	∞
D	(A, D)	1
E	(A, B)	∞

De B para	*Link*	Custo
B	local	0
A	(B, A)	∞
C	(B, C)	1
D	(B, A)	∞
E	(B, E)	1

Figura 2.7: As tabelas de roteamento dos nós A e B após a detecção de falha.

Após a atualização de suas tabelas, os nós A e B propagam essas alterações. O nó A envia o vetor-distância $A = 0, B = \infty, C = \infty, D = 1, E = \infty$ no *link* (A, D), e o nó B envia o vetor-distância $B = 0, A = \infty, C = 1, D = \infty, E = 1$ nos *links* (B, C) e (B, E). Essas mensagens irão disparar novas atualizações por toda a rede até que as tabelas em todos os nós reflitam a nova topologia da rede e, nesse caso, a conectividade da rede será restaurada.

Vale ressaltar que, não é possível efetuar o roteamento de mensagens destinadas a nós cujas entradas na tabela de roteamento possuam custo infinito. Ou seja, durante o período em que as tabelas de roteamento estão convergindo para a nova topologia da rede, os pacotes de dados podem ser descartados. Portanto é desejável que a convergência ocorra no menor tempo possível.

No exemplo da figura 2.2, todos os *links* da rede possuíam o custo 1. Porém, podem existir redes em que os *links* possuam custos diferenciados. Suponha que o *link* (C, E) possua um custo 10, e os outros continuam com o custo 1. Nesse caso a figura 2.8 mostra as entradas correspondentes ao destino C de todas as tabelas de roteamento da rede após a convergência do algoritmo.

Suponha, agora, que o *link* (B, C) falhe, e que esta falha seja detectada pelo nó

De	*Link*	Custo
A para C	(A, B)	2
B para C	(B, C)	1
C para C	local	0
D para C	(A, D)	3
E para C	(B, E)	2

Figura 2.8: As entradas correspondentes ao destino C com o *link* (C, E) possuindo custo 10.

B imediatamente após a ocorrência desse evento. Nesse caso, a entrada na tabela de roteamento do nó B com destino C é atualizada, e por um curto período de tempo irá apresentar um custo ∞.

Considerando que a maioria das implementações utiliza um envio regular de mensagens entre os nós da rede para lidar com as alterações no estado da rede, suponha que o nó A envie uma dessas mensagens, contendo seu vetor-distância, para o nó B. Caso essa mensagem seja recebida por B antes mesmo que a falha detectada tenha sido enviada aos vizinhos de B, o nó B irá adicionar o custo 1 do *link* (A, B) à distância 2 anunciada por A para o nó C; e B nota que a distância 3 é menor que o custo ∞ apresentado em sua tabela de roteamento. Desse modo, o nó B atualiza novamente o custo para alcançar o nó C em sua tabela para 3.

O nó B anuncia a distância 3 aos seus vizinhos A e E, desencadeando novas alterações em suas tabelas de roteamento. A figura 2.9 ilustra a situação das entradas com destino C nesse momento. Pode-se notar que essa situação contempla um ciclo. Pacotes com destino C, roteados por A, serão enviados ao nó B e então retornados, até que o tempo de vida desses pacotes expire. Esse fenômeno é denominado "*bouncing effect*", e apenas irá terminar depois que as tabelas de roteamento convergirem a uma situação coerente.

Nessa situação, as mensagens disparadas pelas mensagens de atualização de A e B geram novas mensagens de atualizações e, a cada rodada, a distância para o nó C é incrementada em 2 nas tabelas de roteamento dos nós A, B, D e E. Isso ocorre até que o custo para C ultrapasse o custo da aresta (C, E), e o nó C anuncie seu vetor-distância no *link* (C, E). Nesse ponto, o nó E atualiza sua tabela de roteamento e anuncia o custo 10 através da aresta (C, E) gerando novas atualizações nos demais nós da rede até que as

De	*Link*	Custo
A para C	(A, B)	4
B para C	(A, B)	3
C para C	local	0
D para C	(A, D)	3
E para C	(B, E)	4

Figura 2.9: As entradas correspondentes ao destino C após o recebimento das atualizações enviadas por B.

tabelas de roteamento apresentem os valores apresentado na figura 2.10.

De	*Link*	Custo
A para C	(A, B)	12
B para C	(B, E)	11
C para C	local	0
D para C	(D, E)	11
E para C	(E, C)	10

Figura 2.10: As entradas correspondentes ao destino C após a convergência do algoritmo.

O período de convergência é muito crítico, pois com a ocorrência de ciclos, pacotes são retransmitidos até expirar o seu tempo de vida, permitindo levar a um congestionamento dos *links*. Esse congestionamento pode causar a perda de pacotes, não somente de dados, mas também de mensagens de roteamento, o que pode atrasar ainda mais a convergência.

Há também outras situações que podem desencadear comportamentos indesejáveis ao roteamento. Um exemplo possível, baseado no exemplo da figura 2.6, é ilustrado pela figura 2.11. Nesse caso os nós A e D ficam isolados da rede. Considerando que a falha no *link* (D, E) ocorra logo após a falha do *link* (A, B), o nó D, ao detectar tal falha, atualiza sua tabela de roteamento conforme mostrado pela figura 2.12.

Analisando apenas o comportamento dos nós A e D, caso D tenha a oportunidade de transmitir o seu vetor-distância ao nó A, A irá atualizar a sua tabela de roteamento e notar que os nós B, C e E são inalcançáveis, e o algoritmo terá convergido. Porém, se A transmitir primeiramente o seu vetor-distância: $A = 0, B = 3, C = 3, D = 3, E = 3$, então o nó D atualizará a sua tabela de acordo com o vetor recebido.

Nesse caso, novamente, é gerado um ciclo no roteamento, pois o nó A enviará mensagens destinadas aos nós B, C e E para o nó D e, por sua vez, o nó D enviará essas men-

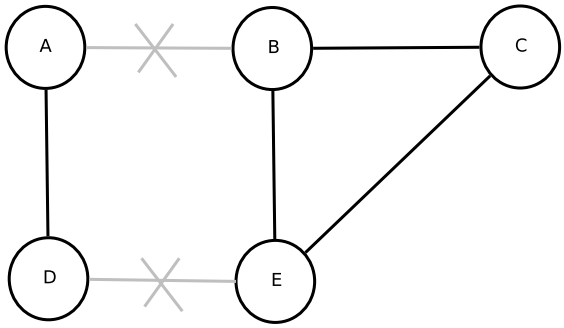

Figura 2.11: Exemplo de falhas que podem desencadear a "contagem ao infinito".

De D para	*Link*	Custo
A	(A, D)	1
B	(D, E)	∞
C	(D, E)	∞
D	local	0
E	(D, E)	∞

Figura 2.12: A tabela de roteamento do nó D após a descoberta da segunda falha.

sagens ao nó A, e vice-versa. A cada atualização das tabelas de roteamento as distâncias para os nós B, C e E irão ser acrescentadas em 2. Esse processo é chamado de "contagem ao infinito (*counting to infinity*)" e, pode ser parado utilizando-se uma convenção para a representação do valor infinito. Esse valor deve ser maior que o valor da distância do pior caminho possível na rede. Quando as distâncias na tabela de roteamento alcançarem esse valor, as entradas são consideradas infinitamente distantes e, portanto, inalcançáveis.

2.3.2 Funcionamento do Algoritmo de Estado de Enlace (Dijkstra)

A família de algoritmos de estado de enlace (*Link-State*) é também conhecida como roteamento pelo caminho mais curto (SPF - *Shortest Path First*) ou algoritmo de Dijkstra, pois baseia-se no algoritmo *Shortest Path First* proposto por E. W. Dijkstra [15]. Por convergir em $O(E * logV)$ iterações, de acordo com [11], o SPF escala melhor que o algoritmo Bellman-Ford, o qual converge em $O(V * E)$, com E sendo o número de arestas e V o número de vértices.

O algoritmo de Dijkstra calcula o caminho mínimo entre um nó s, fonte, e os demais

```
Dijkstra(G, s)
 1. Let D, P ← two arrays with size V, where V is the number of nodes in G
 2. Let L ← a node list
 3. //Initializations
 4. For each node n in G
 5.         D[n] ← ∞
 6.         P[n] ← null
 7. D[s] ← 0
 8. L ← nodes in G
 9. While L is not empty
10.         // Return/Remove the node with smallest distance from L
11.         Let u ← ExtractNode(L)
12.         For each adjacent node v from u
13.             Let new_distance ← D[u] + cost(u, v)
14.             If new_distance < D[v]
15.                 D[v] ← new_distance
16.                 P[v] ← u
```

Figura 2.13: Pseudocódigo do Algoritmo Dijkstra.

nós em um digrafo, G, com pesos não negativos. Ele separa os nós que não foram avaliados em uma lista ordenada de acordo com a distância, do nó fonte, conhecida. Os caminhos mínimos são armazenados em uma estrutura de vetor, na qual cada posição aponta para a posição correspondente ao nó predecessor do caminho até o nó fonte. A figura 2.13 apresenta o pseudocódigo do algoritmo.

Inicialmente o vetor contendo a distância dos nós ao nó fonte, s, é inicializado indicando uma distância infinita em cada nó. Também é inicializado o vetor P, no qual serão armazenados os caminhos mínimos do nó s aos demais nós do grafo. A distância do nó fonte à ele mesmo ($D[s]$) é inicializada como zero e, dentro do *loop* principal, será o primeiro a ser extraído da lista de nós a serem avaliados (L).

Ao avaliar um nó u, são ajustadas as distâncias dos vizinhos deste nó caso seja encontrado um caminho menor passando por u. Quando um nó v tem sua distância atualizada, o caminho desse nó ao nó s também é atualizado, para tanto, apenas é necessário definir

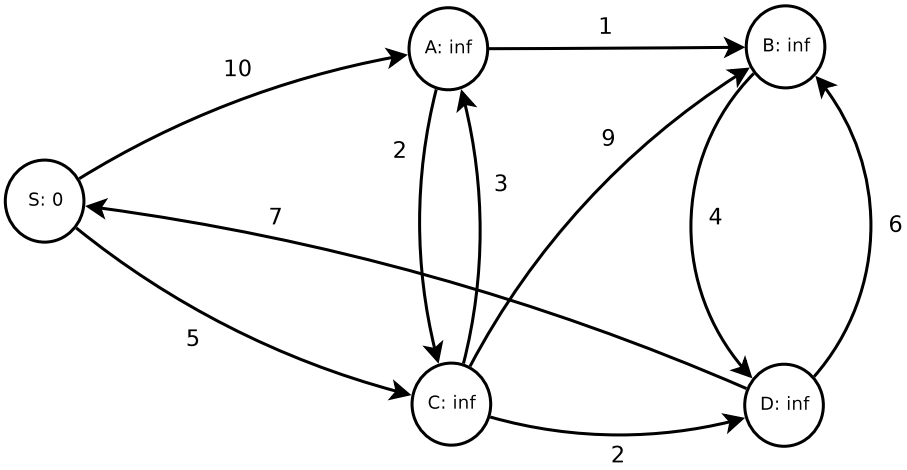

Figura 2.14: Exemplo de Execução do Algoritmo Dijkstra (a).

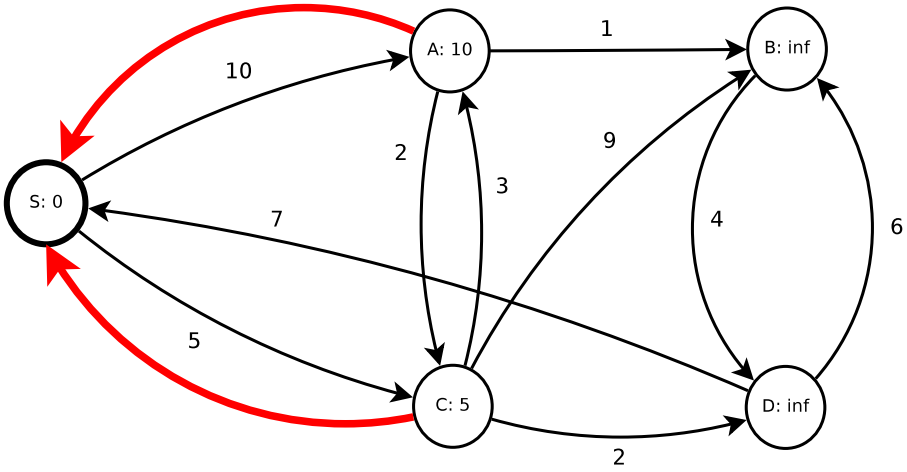

Figura 2.15: Exemplo de Execução do Algoritmo Dijkstra (b).

o nó anterior no caminho como sendo o nó que está sendo avaliado. Quando a lista, L, de nós a serem avaliados estiver vazia o algoritmo terá sido finalizado e as distâncias e caminhos mínimos estarão representados em D e P respectivamente.

Para demonstrar o funcionamento desse algoritmo é utilizado o grafo representado na figura 2.14. Inicialmente, todos os nós possuem uma distância ∞ com exceção do nó fonte (s), cuja distância é zero. Ao avaliar o nó s, conforme mostra a figura 2.15, são alteradas as distâncias dos seus vizinhos, pois é encontrada uma distância com custo menor que infinito. Juntamente com a distância é atualizado o vetor de caminhos mínimos, marcando o nó s como sendo o nó predecessor aos nós A e C.

Nesse ponto, o nó C é escolhido para a avaliação e retirado da lista L, pois é o nó que apresenta a menor distância ao nó s. Então, os nós A, B e D, adjacentes a C, têm

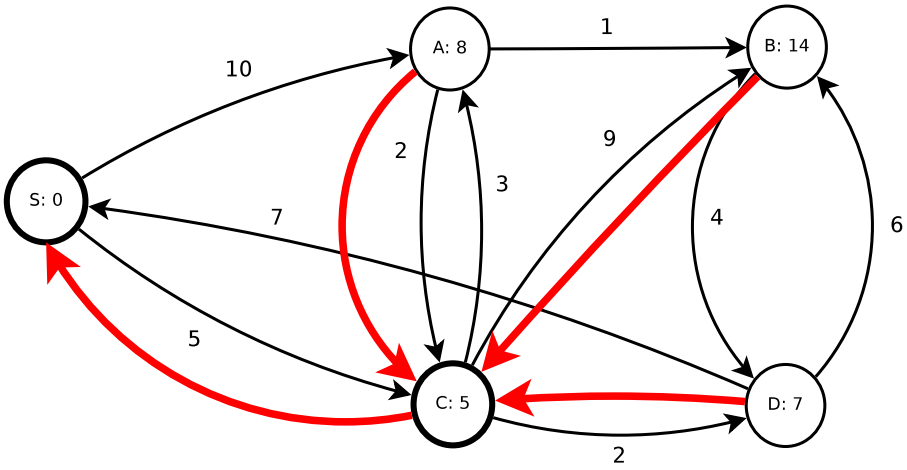

Figura 2.16: Exemplo de Execução do Algoritmo Dijkstra (c).

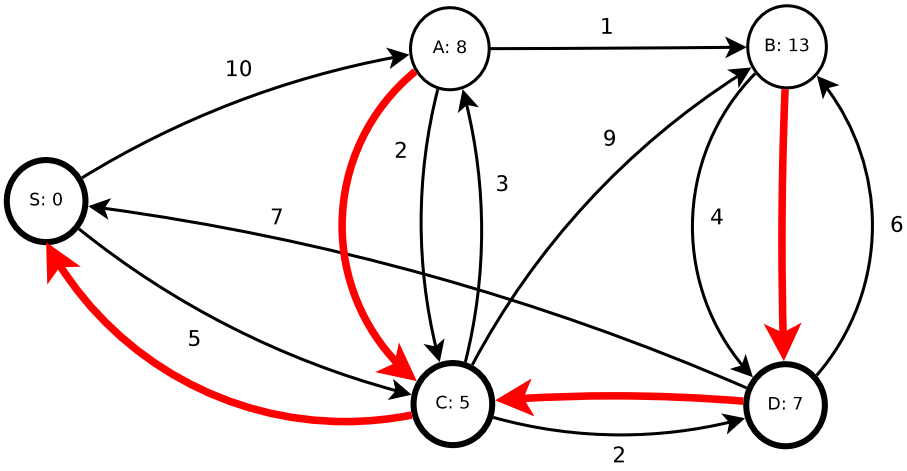

Figura 2.17: Exemplo de Execução do Algoritmo Dijkstra (d).

suas distâncias atualizadas e, o nó C é marcado como predecessor desses nós adjacentes, conforme mostra a figura 2.16. Nas figuras esse exemplo, os círculos em negrito indicam os nós que foram avaliados e as arestas em negrito representam os caminhos mínimos ao nó s.

A figura 2.17 apresenta a situação após o nó D ter sido escolhido para a avaliação. Ao verificar os vizinhos do nó D, apenas as informações de distância e nó predecessor do nó A são atualizadas. Nesse ponto, restam apenas os nós A e B na fila de nós a serem avaliados. Os estados seguintes à avaliação desses nós são apresentados nas figuras 2.18 e 2.19.

Finalmente, após avaliar o nó B, como mostra a figura 2.19, não existem mais nós a serem avaliados na lista L e é terminada a execução do algoritmo. Ao final, o vetor de

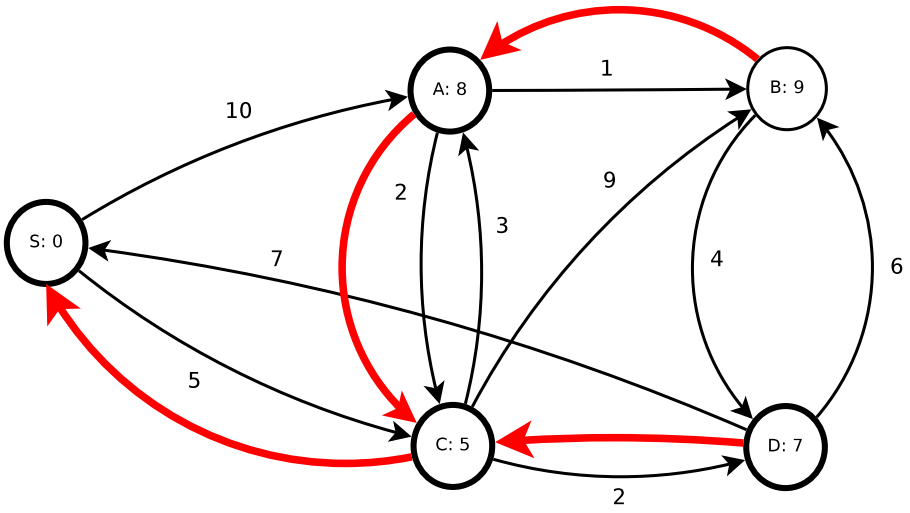

Figura 2.18: Exemplo de Execução do Algoritmo Dijkstra (e).

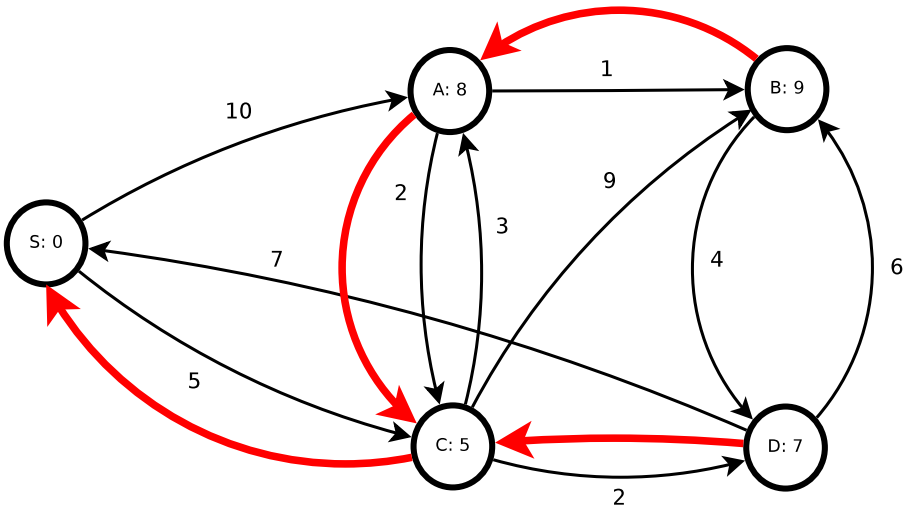

Figura 2.19: Exemplo de Execução do Algoritmo Dijkstra (f).

predecessores apresenta o caminho mínimo de cada nó até o nó fonte s.

2.4 Protocolos de Roteamento Interno

Como mencionado na seção 2.2, um protocolo de roteamento interno (*Interior Gateway Protocol* - IGP) é um protocolo de roteamento que é usando dentro de um AS. Os IGP's podem ser divididos em duas categorias principais: *Distance-Vector Routing Protocols*, que utilizam os algoritmos da família vetor-distância e *Link-State Routing Protocols*, que utilizam os algoritmos da família estado de enlace.

A seguir são apresentados dois dos principais protocolos de roteamento interno, o RIP e o OSPF.

2.4.1 *Routing Information Protocol* (RIP)

O protocolo RIP, documentado inicialmente na RFC-1058 por C. L. Hedricks [27], é fundamentado em uma implementação direta do algoritmo de vetor-distância. Porém, para minimizar alguns dos problemas conhecidos desse algoritmo, como o *bouncing effect* e a contagem ao infinito, descritos na seção 2.3.1, o RIP implementa técnicas como *Split Horizon* e *Triggered Updates* [30] para minimizar esses problemas.

A técnica *Split Horizon* é baseada em uma simples precaução: se um nó A está roteando pacotes, que tem como destino um nó X, pelo nó B, não faz sentido o nó B tentar alcançar o nó X através de A. Desse modo, não faz sentido o nó A anunciar para B a que distância está do nó X. Assim sendo, a mudança no algoritmo original é mínima: ao invés de fazer o *broadcast* do mesmo vetor-distância em todos os *links* de saída, os nós enviam versões diferentes dessa mensagem, considerando que alguns destinos são roteados através desses *links*.

O *Split Horizon* elimina ciclos entre dois nós, pode-se facilmente perceber que essa técnica eliminaria a contagem ao infinito apresentada na seção 2.3.1. Porém, nem todas as formas de ciclos são evitadas pelo *Split Horizon*.

A técnica *Triggered Updates* relaciona-se com a questão de quando enviar o vetor-

distância. Muitas implementações do algoritmo vetor-distância, e no RIP não é exceção, baseiam-se no envio regular do vetor-distância pelos nós da rede. O que permite, ao mesmo tempo, monitorar os vizinhos de cada nó.

Um período muito curto entre o envio dos vetores-distância pode sobrecarregar a rede. Porém, um período muito longo entre os envios pode acarretar no atraso demasiado na sinalização de alterações na topologia da rede. Desse modo, a técnica *Triggered Updates* é uma tentativa de aumentar a eficiência do algoritmo, fazendo com que os nós enviem seus vetores assim que notem alguma alteração em suas tabelas de roteamento, sem ter que esperar até o fim do período.

O protocolo RIP divide os dispositivos participantes em dois modos: ativos e passivos. Os participantes ativos enviam seus vetores para os outros; os participantes passivos apenas "escutam" às mensagens RIP e utilizam-nas para atualizar suas tabelas de roteamento. Um roteador executando o RIP em modo ativo efetua o *broadcast* do seu vetor-distância, tipicamente a cada 30 segundos. O formato do vetor-distância é um conjunto de pares, em que cada par contém um endereço IP (*Internet Protocol*) de um dado *host*, e a distância cuja métrica é a contagem de *hops*.

O RIP define o uso de temporizadores para monitorar os vizinhos de cada nó da rede. Quando um roteador insere uma entrada em sua tabela de roteamento, é iniciado um temporizador para essa entrada. Esse temporizador precisa ser reiniciado sempre que for recebida uma mensagem pela rota indicada na tabela. A documentação de RIP [27], sugere que esse temporizador seja seis vezes o intervalo de transmissão dos vetores, ou seja, tipicamente 180 segundos.

Também é definido na documentação do RIP, o valor 16 para expressar o valor de infinito. Com essa escolha, de um valor pequeno, para expressar o infinito, os problemas de convergência lenta e contagem ao infinito são limitados. Entretanto, o comprimento máximo de uma rota fica limitado em 15.

Uma nova versão do RIP, foi definida por Gary Malkin na RFC-1388 [37] e chamada de RIP-2. O RIP-2 define uma série de aperfeiçoamentos no RIP, como o roteamento em subredes, o suporte a CIDR (*Classless Inter-Domain Routing*) [23], autenticação e

transmissão em *multicast*. A RFC-1388 é completado pela RFC-1387 [38], uma análise do protocolo, e pela RFC-1389 [39], a definição da RIP-2 MIB (RIP-2 *Management Information Base*).

Outra adaptação do RIP é o protocolo RIPng [40]. Esse protocolo foi definido para poder usar o RIP com o protocolo IPv6 (*Internet Protocol version 6*) e apresenta, basicamente, duas diferenças importantes: o uso da segurança IPv6 ao invés das entradas de autenticação do RIP-2; e a adaptação do formato dos pacotes para acomodar os endereços IPv6.

2.4.2 *Open Shortest Path First* (OSPF)

Como dito na seção 2.3.2, o algoritmo SPF possui uma melhor escalabilidade quando comparado ao Bellman-Ford. Por esse motivo, um grupo de trabalho da IETF (*Internet Engineering Task Force*) projetou um protocolo de roteamento interno que usasse esse algoritmo. Esse protocolo foi denominado *Open* SPF (OSPF) [42] e, recomendado pela IAB (*Internet Architecture Board*) como um substituto ao RIP.

Ao invés de calcular as "melhores rotas" de uma forma distribuída, os nós participantes do OSPF executam o cálculo das "melhores rotas" utilizando o conhecimento que possuem sobre a topologia. Esse conhecimento provém de uma base local denominada *Link-State Database*. Cada registro representa um *link* da rede e, contém as informações necessárias para que cada nó possa calcular o caminho mínimo aos demais nós da rede.

O tamanho dessa base de dados, bem como o tempo de computação, a memória necessária e o volume de mensagens do protocolo, aumenta significativamente à medida em que a rede cresce. Para auxiliar a resolução desses problemas, é utilizado um "roteamento hierárquico", *i.e.*, a rede é dividida em um conjunto de partes independentes conectadas por um "backbone". No OSPF, esses conjuntos independentes são chamados de "*areas*", e o "backbone" que conecta esses conjuntos é chamado de "*backbone area*". Em cada uma dessas áreas, os nós incluem em sua tabela de roteamento apenas os estados dos *links* da área específica, de forma que o custo do protocolo passa a ser proporcional apenas ao tamanho da área escolhida para o nó [30].

O algoritmo SPF é usado para calcular o caminho mínimo do nó que executa o processo do OSPF para cada destino possível. Desse cálculo, o nó obtém o próximo *hop* para cada destino e, após montar a tabela de roteamento, o processo OSPF passa a informação para que o protocolo execute o roteamento. Cada *link* pode possuir várias métricas. As rotas são, inicialmente, calculadas de acordo com a métrica padrão. Se o nó for capaz de efetuar o roteamento baseado em Tipo de Serviço (ToS - *Type of Service*), ele deve calcular também as rotas de acordo com cada métrica existente.

Os roteadores OSPF comunicam-se através do protocolo OSPF. Esse protocolo é executado diretamente sobre o IP e, é de fato composto de outros três sub-protocolos: Hello, Exchange e Flooding. O protocolo Hello é usado para dois propósitos: verificar se os *links* estão operacionais; e eleger um nó "designado" e de um nó "*backup*", os quais são utilizados em redes com conectividade total para minimizar o *overhead* do OSPF. O protocolo Exchange realiza a sincronização inicial entre as bases de dados de dois nós que tenham estabelecido uma comunicação entre si. O protocolo Flooding é responsável por manter essas bases de dados sincronizadas [30].

2.5 Roteamento Externo - *Border Gateway Protocol* (BGP)

Cada AS precisa configurar um ou mais de seus roteadores para comunicarem-se com outros AS's. Um roteador é configurado para saber, ou coletar, informações sobre as redes pertencentes ao AS. Essas informações devem ser enviadas para os outros AS's e, devem ser aceitas informações de outros AS's e disseminadas internamente. Essas informações são propagadas entre os diversos AS através de algum dos protocolos de roteamento externo (*Exterior Gateway Protocols*). Esse termo denota a classe de protocolos utilizados para trocar informações entre AS's.

O protocolo EGP (*Exterior Gateway Protocol*) [41] foi um dos primeiros protocolos desenvolvidos para permitir a troca de informações entre AS's distintos. Com o crescimento da Internet, as limitações do EGP se tornaram inaceitáveis e, foi padronizado o BGP, como substituto ao EGP, pelo *Border Gateway Protocol Working Group* da IETF.

O BGP é o protocolo usado para a troca de informações na Internet atualmente. Ele

tem evoluído através de quatro versões. A primeira versão foi publicada em 1989 como a RFC-1105 [34]. Na sequência foram publicadas as versões 2, 3 e 4 como as RFC's: RFC-1163 [35], RFC-1267 [36] e RFC-1654 [46], sendo que a quarta versão ainda foi revisada em 1995 como a RFC-1771 [47].

O BGP utiliza um algoritmo de roteamento conhecido como vetor-caminho (*path vector*) [30]. Esse algoritmo baseia-se no algoritmo vetor-distância, porém ao invés de transmitir as distâncias propriamente ditas, as informações de atualizações são transmitidas contendo uma lista completa das redes, ou AS's, percorridos entre uma fonte e um destino. Esse fator possibilita a prevenção de ciclos em topologias complexas.

Apesar de ser o protocolo para roteamento externo utilizado universalmente na Internet, o BGP apresenta problemas, entre eles o da convergência lenta, examinado na próxima seção.

2.6 Trabalhos Relacionados

Esta seção contempla novas estratégias que visam otimizar a política de roteamento na Internet. Inicialmente, são apresentadas algumas propostas para o aperfeiçoamento dos principais protocolos de roteamento interno e externo, OSPF e BGP. Em seguida, são descritos trabalhos que propõem novos algoritmos para o roteamento IP tolerante a falhas. Por fim, são apresentados alguns trabalhos focados no roteamento baseado em QoS (*Quality of Service*) [12], mais especificamente no MPLS (*Multi-Protocol Label Switching*) [9].

A latência de convergência do protocolo utilizado para roteamento externo na Internet, o BGP, é de aproximadamente três minutos, sendo que já foram observados picos de até quinze minutos [32]. A inconsistência das tabelas de roteamento nos nós de uma rede durante esse período de tempo pode causar a perda de pacotes e a interrupção de conexões estabelecidas.

Já no OSPF, um dos principais protocolos de roteamento interno, essa latência de convergência é superior a 40 segundos, como demonstrado a seguir. De acordo com a especificação do OSPF [42], após cessar de escutar as mensagens de *Hello* de um roteador

por um certo período de tempo, esse roteador é marcado como falho. O timer para esse período de tempo, na especificação do OSPF é sugerido como 4 vezes o tempo de transmissão de uma mensagem de *Hello*, que é tipicamente de 10 segundos. Ou seja, são necessários tipicamente 40 segundos para um roteador detectar a falha em um roteador vizinho. A latência de convergência ainda compreende o tempo de disseminação da informação sobre a falha na rede e o tempo necessário para cada roteador recalcular sua tabela de roteamento. Visando diminuir essa latência de convergência do BGP e do OSPF, foram propostas e implementadas diversas soluções. Algumas dessas soluções são descritas a seguir.

Em [2], Basu e Riecke avaliam o funcionamento do OSPF em três diferentes condições: utilizando extensões para engenharia de tráfego, utilizando o envio de mensagens de *Hello* em tempos inferiores a um segundo e utilizando estratégias alternativas para a atualização das informações sobre o estado dos enlaces. Essa avaliação considera a latência de convergência da rede, a utilização dos processadores e o número de mudanças nas tabelas de roteamento. Considerando especificamente a utilização de envio de mensagens de *Hello* em tempos inferiores a um segundo, os resultados da avaliação apresentam uma considerável redução na latência de convergência. Para intervalos superiores a 500 milissegundos, essa redução na latência de convergência é obtida sem que haja um impacto significativo nos outros dois fatores de estabilidade considerados.

Pei *et. al.* [45] propõem um novo mecanismo, chamado BGP-RCN (*BGP with Root Cause Notification*, que limita superiormente o tempo de convergência do BGP em $O(d)$, onde d é o diâmetro da rede medido pelo número de *hops* de AS. Esse mecanismo consiste na inclusão de informações sobre a causa específica da geração de uma mensagem de atualização de topologia dentro dessa mensagem. Uma vez que um nó receba a primeira mensagem de atualização gerada por uma falha em um *link*, esse nó pode evitar de usar quaisquer caminhos que tenham se tornado obsoletos devidos à mesma falha.

Com o mesmo intuito de diminuir o tempo de convergência do BGP, Bremler-Barr *et. al.* [6] sugerem uma pequena mudança no protocolo, permitindo a geração e propagação "mensagens de retirada" (*withdrawal messages*). Uma "mensagem de retirada" informa

a um nó vizinho que um caminho anunciado previamente a um AS não é mais válido. As "mensagens de retirada" são propagadas assim que geradas ou recebidas, sem atraso. As demais mensagens de roteamento continuam a propagar com a mesma periodicidade em que são definidas no padrão. Ou seja, as mensagens que informam falhas na rede são propagadas rapidamente, e as mensagens que informam as recuperações ou novas conexões na rede são propagadas mais lentamente.

Feigenbaum *et. al.* [21] propõem um mecanismo de roteamento, baseado no BGP, que utiliza uma técnica conhecida como *mechanism design* e desenvolvida, inicialmente, para jogos computacionais. Essa técnica foi adaptada para utilização em roteamento por Nissan e Hershberger [43, 28]. Nesse trabalho, ao invés de *links*, são os nós a serem tratados como agentes estratégicos e, ao invés de calcular os caminhos de menor custo (*Lowest-Cost Pahts* para um par origem-destino, são calculados caminhos de menor custo para todos os pares origem-destino possíveis. Esse mecanismo é calculado como um algoritmo distribuído, como uma extensão direta ao BGP. Contrastando com os algoritmos usados em [43, 28], esse mecanismo causa apenas um pequeno acréscimo no tamanho das tabelas de roteamento e no tempo de convergência.

Além dessas alternativas propostas para o aperfeiçoamento do BGP e do OSPF, existem diferentes estratégias que se utilizam de novas abordagens para melhorar o roteamento na Internet. A seguir são descritas algumas dessas abordagens.

Em [17] e [49], é proposto o uso de rotas alternativas, chamadas *detours*, em casos de falha no roteamento. Os *detours* são selecionados de acordo com a conectividade dos nós da rede. Quanto maior a conectividade de um nó selecionado para efetuar o *detour*, maior a probabilidade desse *detour* funcionar corretamente. Essa abordagem não leva em consideração a conectividade dos demais nós da rota alternativa.

Echenique *et. al.* [18] sugerem o uso de algoritmos que apresentam um melhor desempenho quando comparados ao algoritmo padrão de roteamento entre AS's. Esses algoritmos utilizam tanto informações da topologia quanto informações de tráfego da rede. O algoritmo envolve a troca constante de mensagens de monitoração de tráfego entre os roteadores da rede.

Outro trabalho que leva em consideração o tráfego da rede é proposto por Yan *et. al.* [51]. Nesse trabalho é citada uma estratégia para melhorar a eficiência do transporte em redes complexas, como a Internet. Ao invés de apenas considerar o caminho mínimo como estratégia de roteamento, é proposto um algoritmo que encontra o chamado caminho "eficiente" (*efficient path*), que considera a possibilidade de congestionamento ao longo dos caminhos escolhidos para o roteamento. Essa abordagem visa redistribuir o tráfego que passa pelos nós centrais, aqui expressos como nós de maior grau, para nós que possuam um menor grau. Porém para realizar essa tarefa, cada roteador precisa ter o conhecimento completo da topologia da rede. Situações de falhas ou alterações constantes na topologia podem prejudicar o algoritmo.

Wei Li [33] propõe uma nova arquitetura para o roteamento externo. Essa arquitetura, denominada REN (*Routing Enhancement Network*), utiliza uma rede *overlay* [1] sobre a arquitetura original de roteamento. A arquitetura REN, além da rede *overlay*, utiliza uma série de mecanismos para aprimorar o roteamento. Dentre esses mecanismos, pode-se citar a limitação do tamanho das tabelas de roteamento através de normas de restrição aplicadas aos roteadores centrais e a não propagação de prefixos de redes pequenas dentro da rede *overlay*.

Existe um conjunto emergente de tecnologias, denominado *IP Fast Reroute (IPFRR)* [24], que tem por objetivo reparar rapidamente as condições de falha em redes IP. A idéia principal do IPFRR é efetuar o reparo através do re-roteamento do tráfego através de alternativas pré-computadas. A seguir são apresentadas algumas das principais soluções propostas nesse conjunto.

Quando um *link* ou nodo falha, apenas os vizinhos dessa falha possuem, inicialmente, conhecimento de sua ocorrência. Em uma rede que usa IPFRR, os roteadores que são vizinhos dessa falha tentam repará-la. Esses roteadores devem enviar os pacotes para os seus destinos mesmo sem os outros roteadores da rede terem o conhecimento da ocorrência dessa falha. Para resolver esse problema, em [7] é proposto um mecanismo de encapsulamento desses pacotes para endereços que identifiquem o componente falho a ser evitado. Esses endereços, denominados *not-via*, são atribuídos aos roteadores da rede. A semântica

de um endereço *not-via* é que um pacote endereçado a esse endereço deve ser entregue ao roteador que anuncia esse endereço, porém, não deve ser entregue via o elemento ao qual o endereço está associado. Por exemplo, um pacote destinado a um roteador X que possui um endereço *not-via* Xa deve ser entregue ao roteador X sem passar pelo elemento a. Para utilizar esses endereços é necessário que cada roteador calcule previamente o caminho que usaria na ocorrência de qualquer falha possível na rede. Para calcular isso, cada roteador falha todos elementos da rede, um de cada vez, e calcula sua melhor rota para os vizinhos do elemento falho utilizando o endereço *not-via* desses vizinhos.

Kvalbein *et. al.* [31] apresentam um novo esquema para o tratamento de falhas em redes IP. A idéia central desse esquema, chamado de *Multiple Routing Configurations (MRC)*, consiste em utilizar a topologia conhecida da rede e construir um conjunto de configurações de roteamento. Uma configuração é definida como um conjunto de pesos associados aos *links* da topologia. Em uma configuração dita resistente a falha em um *link* l e em a um nodo n, são associados pesos aos *links* da topologia de tal forma que o tráfego nunca é roteado através do *link* l e do nodo n. O conjunto de configurações é criado de modo com que cada componente da rede fique isolado em uma configuração distinta. Para cada configuração é executado um algoritmo similar ao OSPF para criar as tabelas de roteamento desse conjunto de configurações em cada nodo. Permitindo que na ocorrência de uma falha seja usada uma configuração que isola o elemento falho para continuar o processo de roteamento.

Tarik *et. al.* [19] apresentam um mecanismo aprimorado para o re-roteamento IP (*IP Fast Rerouting*). Esse trabalho aprimora o no trabalho proposto em [31]. A principal diferença para com o MRC [31] consiste na não obrigatoriedade do isolamento de cada *link* em uma das topologias construídas. Isso propicia uma melhora em dois pontos: é mais fácil para se construir e gerenciar múltiplas topologias, e o roteamento em cada topologia se torna menos restrito.

Múltiplos serviços de tempo real estão sendo oferecidos através da Internet. E, esses serviços requerem que a rede ofereça certas garantias de funcionamento, como prover uma largura de banda desejada ou propiciar um atraso limitado na entrega dos pacotes aos

destinatários. Entretanto, os atuais protocolos de roteamento na Internet são projetados de acordo com o modelo de serviços melhor-esforço. Nesse modelo, todas as transmissões de dados são concorrentes entre si, não há distinção entre os pacotes de dados.

O roteamento baseado em QoS [12] busca atender às necessidades requeridas pelos serviços de tempo real. Para prover QoS na Internet, muitas técnicas foram propostas e estudadas, entre elas: *Integrated Services (IntServ)* [5], *Differential Services (DiffServ)* [4] e *MultiProtocol Label Switching (MPLS)* [9]. A seguir são apresentados dois trabalhos que visam aperfeiçoar o roteamento MPLS, para permitir a construção de redes MPLS confiáveis mesmo na presença de falhas.

Calle *et. al.* [8] sugerem o aperfeiçoamento do roteamento MPLS através da utilização de um novo paradigma para a "proteção" da rede (NPD - *Network Protection Degree*). Esse paradigma sugere o uso de novas estratégias de proteção para a construir redes MPLS confiáveis. O NPD consiste na avaliação de dois aspectos relativos à ocorrência de falhas na rede. A primeira avaliação está relacionada ao grau de sensibilidade a falhas (FSD - *Failure Sensibility Degree*), e mede a probabilidade da ocorrência de uma falha na rede. A segunda avaliação está relacionada ao grau de impacto de falhas (FID - *Failure Impact Degree*), e determina o impacto de uma falha sobre a rede.

Para tornar viável a recuperação em redes MPLS e propiciar um serviço confiável, Huang *et. al.* [29] propõem um novo mecanismo para a proteção de caminhos (*path protection mechanism*). Esse mecanismo é baseado na utilização de uma estrutura de notificação reversa (*reverse notification tree structure*), que torna mais eficiente e rápida a distribuição de mensagens de notificação de falhas.

Capítulo 3

Roteamento Baseado em Avaliação de Fluxo Máximo

O objetivo desse capítulo é apresentar a seleção de rotas baseada em avaliação de fluxo máximo, originalmente introduzida pela avaliação de critérios de conectividade em [17]. As principais características do algoritmo de roteamento baseado em critérios de conectividade [50] são descritas a seguir.

O algoritmo de roteamento recebe como entrada um par de nós da rede, correspondentes à origem e ao destino de uma mensagem a ser enviada. O algoritmo é executado em cada nó, iniciando pelo nó de origem e determinando o próximo nó da rota dentre os nós vizinhos. Quando a mensagem chega ao nó escolhido, esse executa o mesmo procedimento para escolher o nó seguinte, até que a mensagem chegue ao nó de destino.

O algoritmo é tolerante a falhas, pois baseia-se em critérios de conectividade para selecionar rotas robustas que possibilitem, na ocorrência de falhas, a utilização de um outro caminho. Esse caminho, ou desvio, parte do ponto em que a falha é conhecida, em direção ao destino final da mensagem.

Esse algoritmo é dinâmico, pois cada nó, ou roteador, é responsável por determinar apenas o próximo passo, *hop*, das mensagens a serem roteadas. Ou seja, um caminho é construído dinamicamente pelos roteadores intermediários desse caminho. A adoção dessa abordagem implica em um algoritmo com maior robustez, pois as alterações na topolo-

gia da rede são, comumente, refletidas mais rapidamente em roteadores mais próximos. Então, conforme uma mensagem está sendo roteada, cada roteador mais próximo ao destino final tem uma representação mais atualizada da topologia da rede naquele ponto, podendo assim efetuar a escolha de uma rota mais adequada.

O restante do capítulo está organizado da seguinte forma. A seção 3.1 define os conceitos de fluxo máximo e corte mínimo utilizados nesse trabalho. A seção 3.2 descreve a seleção de rotas baseada na avaliação conjunta do fluxo máximo e caminho mínimo. Em seguida, a seção 3.3 apresenta a especificação formal do algoritmo usado como base do trabalho e ilustra seu funcionamento. Por fim, é apresentada uma avaliação da complexidade desse algoritmo.

3.1 Definições - Fluxo Máximo e Corte Mínimo

Esta seção apresenta algumas definições relacionadas ao problema de fluxo em redes. Essas definições serão utilizadas nas próximas seções para especificar os critérios de seleção de rotas utilizadas pelo algoritmo.

Definição 3.1 (Fluxo): Um fluxo em uma rede X é uma função f que assinala um número real para cada aresta da rede: $f : VxV \rightarrow \Re$, de acordo com as seguintes propriedades:

- *Restrição de capacidade:* $\forall u, v \in V, f(u,v) \leq c(u,v)$
- *Simetria oblíqua:* $\forall u, v \in V, f(u,v) = -f(v,u)$
- *Conservação de Fluxo:* $\forall u \in V - \{s,t\}$:

$$\sum_{v \in V} f(u,v) = 0$$

Definição 3.2 (Redes Residuais): Seja $G = (V,E)$ um grafo; seja $f(u,v)$ um fluxo em G. A capacidade residual c_f é a quantidade de fluxo adicional que pode passar por

(u, v) sem exceder a capacidade $c(u, v)$:

$$c_f(u, v) = c(u, v) - f(u, v)$$

Dada a rede $G = (V, E)$ e um fluxo $f(u, v)$, a rede residual de G induzida por f é denotada por $G_f = (V, E_f)$, onde:

$$E_f = \{(u, v) \in V \times V | c_f(u, v) > 0\}$$

Ou seja, cada aresta da rede residual pode admitir um fluxo maior que zero.

Definição 3.3 (Caminhos em Ampliação): Dada a rede $G = (V, E)$ e um fluxo f, um caminho em ampliação p é um caminho simples de s a t na rede residual G_f. A quantidade máxima pela qual o fluxo pode ser aumentado em um caminho em ampliação p é chamada de capacidade residual de p, e é expressa por:

$$c_f(p) = \boldsymbol{min}\{c_f(u, v) | (u, v) \in p\}$$

Então, a capacidade residual de um caminho em ampliação p corresponde à menor dentre as capacidades das arestas que fazem parte de p.

Definição 3.4 (Cortes de Fluxo): Seja $G = (V, E)$ uma rede; um corte (S, T) de um fluxo em rede é uma separação do conjunto de vértices V em dois conjuntos S e $T = V - S$, de forma que a origem $s \in S$ e o depósito $t \in T$. Se f é um fluxo, então o fluxo líquido pelo corte (S, T) é definido como $f(S, T)$. A capacidade do corte (S, T) é $c(S, T)$. Um corte mínimo em uma rede é um corte cuja capacidade é mínima dentre todos os cortes da rede.

O fluxo líquido por qualquer (S, T) corte é o mesmo, e é igual ao valor do fluxo máximo. Como conseqüência disso, pode-se afirmar que o fluxo máximo em uma rede é limitado superiormente por um corte mínimo da rede. Este resultado decorre do teorema abaixo.

Teorema de Fluxo Máximo e Corte Mínimo: Se f um fluxo; seja $G = (V, E)$ um fluxo em rede com origem s e depósito t, então as condições a seguir são equivalentes:

- f é um fluxo máximo em G
- A rede residual G_f não contém nenhum caminho em ampliação
- $|f| = c(S, T)$ para algum corte (S, T) de G.

Este teorema foi provado por P. Elias, A. Feinstein, e C.E. Shannon em 1956, e, independentemente, por L.R. Ford, Jr. e D.R. Fulkerson [22] no mesmo ano, informando que o valor de um fluxo máximo é de fato igual à capacidade de um corte mínimo.

Um algoritmo clássico para a avaliação do fluxo máximo, ou corte mínimo, é o algoritmo de Ford e Fulkerson [22, 11] que é utilizado pelo algoritmo de roteamento proposto em [50] e descrito a seguir.

3.2 Roteamento Baseado em Múltiplos Critérios

Para a selecionar o próximo nó da rota de uma mensagem, este algoritmo avalia as arestas adjacentes ao nó em que é executado. Essa avaliação é efetuada considerando critérios específicos para mensurar as características pertinentes de roteamento. A aresta que apresentar o melhor compromisso entre esses critérios é selecionada para o roteamento.

É possível definir um peso parametrizável para cada critério de avaliação. Esse peso é utilizado para possibilitar um ajuste que enfatize um, ou mais, critérios. No algoritmo apresentado é descrita a utilização de dois critérios: um critério para avaliar a redundância de caminhos entre o nó correspondente à aresta avaliada e o nó de destino, representado pela cardinalidade do fluxo máximo (ou corte mínimo); e um critério que avalia a distância entre esses nós, isto é, o comprimento do menor caminho. Porém, é possível generalizar o conceito de critérios de avaliação para abranger critérios baseados em outras métricas como, por exemplo, o congestionamento da rede.

Os critérios para a avaliação das arestas são funções que, a partir de uma aresta, retornam um valor numérico. Dessa forma, a equação correspondente ao cálculo da avaliação

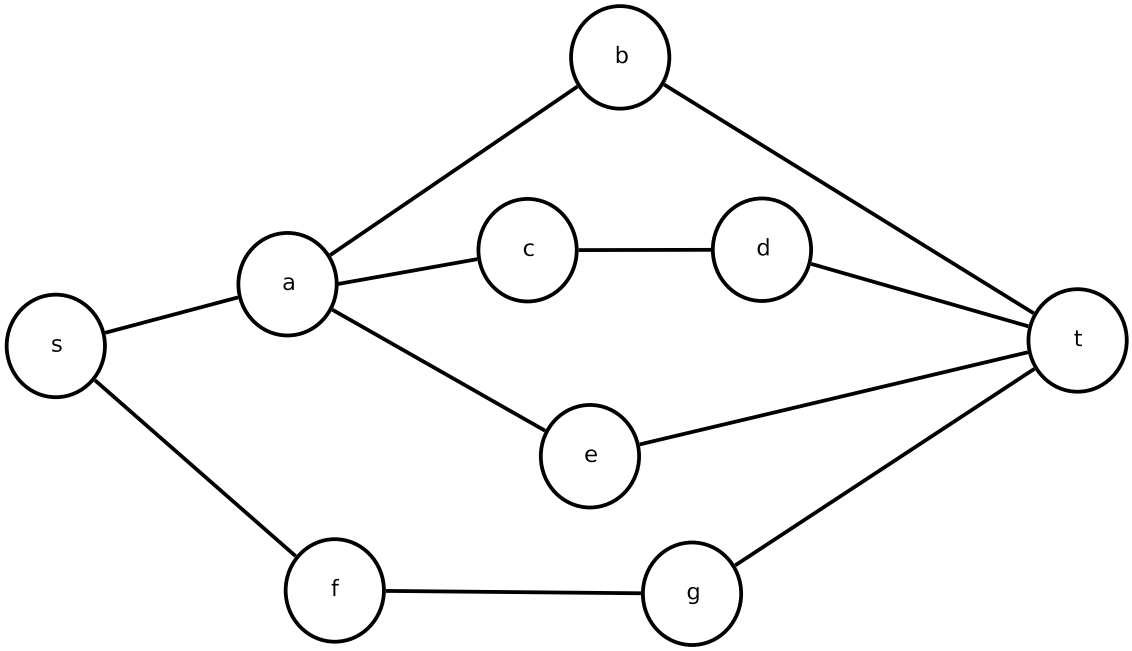

Figura 3.1: A representação de uma rede por um grafo.

das arestas pode ser representada por:

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

Nessa expressão, a função $\Gamma(G, e)$ é a função de avaliação (*trade-off*), e é a aresta sendo avaliada, C é o conjunto de critérios e w_n é o peso que está associado ao critério c_n. A aresta que possuir o maior valor para $\Gamma(G, e)$ será a aresta escolhida para o roteamento da mensagem.

Para exemplificar a avaliação das arestas adjacentes a um nó, através de critérios de avaliação, é apresentada a figura 3.1 que representa uma rede, tendo s como origem e t como destino.

Considerando que, inicialmente, o algoritmo esteja sendo executado no nó s e que s deva enviar uma mensagem à t; as arestas (s, a) e (s, f) são avaliadas de acordo com a função $\Gamma(G, e)$.

Porém, o grafo utilizado na avaliação desconsidera dois conjuntos de arestas do grafo original: arestas percorridas pela mensagem em roteamento; e arestas adjacentes ao nó que executa o processo de roteamento. Desse modo, para avaliação das arestas (s, a) e

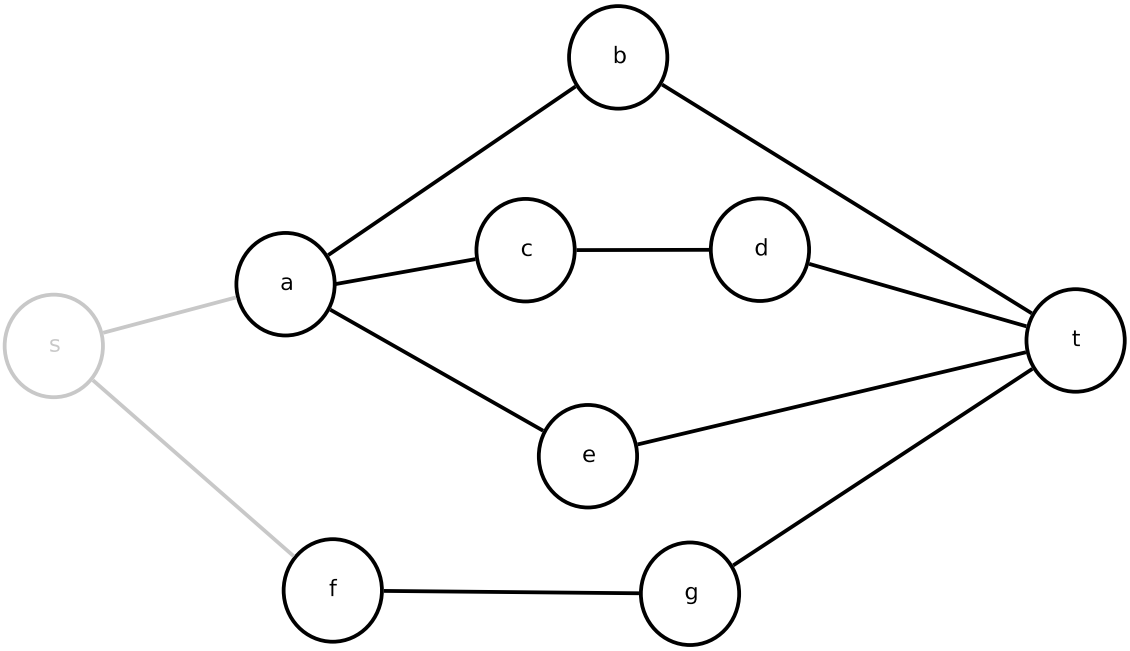

Figura 3.2: A representação do grafo utilizado para a avaliação das arestas iniciais em uma rede.

(s, f) é utilizado o grafo representado na figura 3.2.

Para a avaliação da aresta $e = (s, a)$ a função $\Gamma(G, e)$, o nó s calcula os critérios de fluxo máximo e caminho mínimo considerando o nó a como origem e o nó t como destino.

O cálculo do fluxo máximo, que é a principal contribuição desse algoritmo, é efetuado utilizado o algoritmo clássico de Ford-Fulkerson [22, 11], representado pela figura 3.3. Sua execução utiliza uma estrutura de grafo valorado $G = (V, E)$, em que cada aresta $e \in E$ possui uma capacidade c, e uma estrutura auxiliar de grafo residual $G_f = (V, E_f)$.

```
Ford-Fulkerson(G, s, t)
```

1. For each edge (u, v) in $E[G]$
2. Let $f(u, v) \leftarrow 0$
3. Let $f(v, u) \leftarrow 0$
4. While there is a path p from s to t in G_f, such that $c_f(u, v) > 0, \forall (u, v) \in p$
5. Find $c_f(p) = \textbf{\textit{min}}\{c_f(u, v) | (u, v) \in p\}$
6. For each edge $(u, v) \in p$)
7. Let $f(u, v) \leftarrow f(u, v) + c_f(p)$
8. Let $f(v, u) \leftarrow f(v, u) - c_f(p)$

Figura 3.3: O algoritmo de Ford-Fulkerson.

O algoritmo de Ford-Fulkerson baseia-se na adição de capacidades residuais $c_f(p)$, de caminhos em ampliação p encontrados, ao fluxo f estabelecido. O fluxo máximo é alcançado quando não existirem mais caminhos em ampliação no grafo residual.

Quando são consideradas capacidades com valores que não sejam inteiros, não se pode ter certeza de que o algoritmo de Ford-Fulkerson termine sua execução, apenas pode-se garantir que a resposta é correta caso o algoritmo termine. Porém, quando são consideradas apenas capacidades com valores inteiros, o algoritmo sempre termina, pois, o fluxo aumenta no mínimo em uma unidade a cada caminho em ampliação encontrado, o que consegue-se, por exemplo, através de uma busca em largura que possui complexidade $O(E)$. Quando são considerados valores inteiros a complexidade do algoritmo é $O(E*|f|)$, com E sendo o número de arestas do grafo e $|f|$ o fluxo máximo.

Como no algoritmo de roteamento o fluxo máximo não ultrapassa o grau do nó de origem (as arestas são consideradas como tendo apenas capacidade igual a um) e o grau do nó é, no pior caso, $O(V)$, pode-se afirmar que a complexidade do cálculo do critério de fluxo máximo é $O(V * E)$ para cada nó avaliado, com V sendo o número de nós do grafo.

Na avaliação do primeiro critério, fluxo máximo entre os nós avaliados e o destino, para a aresta (s, a) é encontrado um valor de fluxo máximo igual a 3, ou seja, existem três caminhos disjuntos do nó a ao nó t. Como o foco do algoritmo é a tolerância a falhas, são sugeridos valores positivos relativamente altos como peso para esse critério.

O cálculo do segundo critério, comprimento do caminho mínimo, é efetuado utilizando o algoritmo de Dijkstra [15]. Como as arestas são consideradas tendo capacidade igual a um, a execução desse algoritmo é equivalente a uma busca em largura, portanto, de complexidade $O(E)$.

Na avaliação do segundo critério para a aresta (s, a) seria encontrado o valor 2, correspondendo ao comprimento do menor caminho entre o nó a e f. Nesse caso, basta considerar o primeiro caminho encontrado pela busca em largura, por exemplo (a, e, t). Como quanto maior o comprimento do caminho mínimo, maior é o valor desse critério, o peso correspondente a esse critério deve ser negativo.

3.3 Especificação e Exemplificação do Algoritmo de Roteamento

Nessa seção é apresentada, primeiramente, uma especificação do algoritmo e, em seguida, são considerados casos para a exemplificação de seu funcionamento.

3.3.1 Especificação do Algoritmo

O modelo de roteamento proposto por esse algoritmo é do tipo *hop-by-hop*. Nesse modelo a execução do algoritmo em cada nó retorna apenas a próxima aresta do caminho até destino, como no algoritmo de Bellman-Ford. Uma das principais vantagens desse modelo é propiciar o funcionamento do algoritmo mesmo que a topologia conhecida não reflita as condições reais da rede.

Porém, para que seja factível o roteamento das mensagens sem que a topologia conhecida pelos nós da rede esteja atualizada, a escolha das arestas utilizadas no roteamento deve ser efetuada cuidadosamente, de forma a evitar ciclos (*loops*), aumentar a confiabilidade e a robustez e reduzir o comprimento da rota sempre que possível. Grande parte dos algoritmos de roteamento efetua essa escolha utilizando o comprimento dos caminhos como única ou principal métrica para a seleção do melhor caminho. Porém, aplicações críticas podem preferir utilizar uma rota mais longa, caso a confiabilidade e/ou robustez sejam maiores para essa rota.

Nesse algoritmo, o comprimento dos caminhos é utilizado como critério secundário para a seleção do melhor caminho. O principal critério para essa seleção é o corte mínimo, ou fluxo máximo, entre o nó avaliado e o destino [22]. Esse critério é utilizado para avaliar a redundância dos caminhos, visto que, quanto maior o fluxo máximo, maior a quantidade de caminhos disjuntos e, conseqüentemente, maior é o número de desvios que podem ser utilizados em caso de falha.

Desse modo, a seleção de cada aresta da rota de uma mensagem é efetuada com base em cálculo de compromisso, *trade-off*, descrito na seção 3.2. Esse cálculo avalia a redundância de caminhos e o comprimento do menor caminho propiciados pela escolha de uma aresta. Após a execução desse cálculo para cada nó adjacente ao nó que executa o

processo de roteamento, a aresta que leva ao nó com a melhor avaliação é escolhida para o roteamento.

O cálculo das métricas de seleção do algoritmo pressupõe que cada nó possui uma representação local da topologia da rede. Essa representação é feita através de uma estrutura de grafo direcionado, contendo um conjunto de nós e um conjunto de arestas. Tal estrutura é atualizada através de troca de mensagens periódicas entre os nós da rede.

A rede ilustrada na figura 3.1 pode ser utilizada para exemplificar a representação da topologia da rede em cada nó. Considerando que a rede não apresente falhas e que já tenha decorrido o tempo de convergência do algoritmo, em cada nó da rede haverá uma representação da topologia com os seguintes conjuntos: o conjunto de nós $V - \{s, a, b, c, d, e, f, g, t\}$ e o conjunto de arestas $E - \{(s, a), (a, s), (s, f), (f, s), (a, b), (b, a), (a, c), (c, a), (a, e), (e, a), (b, t), (t, b), (c, d), (d, c), (d, t), (t, d), (e, t), (t, e), (f, g), (g, f), (g, t), (t, g)\}$.

A precisão do algoritmo está diretamente relacionada com a correção dessa lista. Quando ocorrem mudanças na topologia da rede, os nós enviam mensagens de atualização de topologia aos nós vizinhos. Quando um nó recebe uma mensagem de atualização, ele atualiza a representação local da topologia da rede, incluindo ou removendo as arestas indicadas na mensagem.

Essas mensagens são enviadas mesmo quando não ocorrem alterações na topologia da rede, tendo como finalidade a verificação da funcionalidade da aresta. Caso nenhuma mensagem seja recebida após um período de tempo, *time-out*, a aresta é considerada falha, e essa informação 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 essa informação é marcada para ser enviada a outros nós.

A figura 3.4 mostra a parte do código que é responsável por manter a topologia em cada nó da rede. Cada nó inicia a execução do procedimento `TopologyMaintainer(s)` no momento em que entra na rede e a cada α segundos, com α podendo ser parametrizável, envia uma mensagem de atualização para todos os seus vizinhos. Cada mensagem de atualização contém apenas informações sobre alterações na topologia da rede. Essas

```
TopologyMaintainer(s)
```

1. Let $s.KnownTopology \leftarrow$ a graph containing only s
2. Let $s.TimeStamp \leftarrow 1$
3. Let $s.LastEdgeInformation \leftarrow$ 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 $i \leftarrow < l, FALSE, s.TimeStamp >$
9. Increment $s.TimeStamp$
10. Remove l from $s.KnownTopology$
11. Let $s.LastEdgeInformation[l] \leftarrow i$
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 $payload \leftarrow s.EdgeInformationToSend[l]$
16. Let $t \leftarrow$ the node for which l points
17. Run `SendMessage(payload, s, t, TRUE)`

Figura 3.4: Especificação do processo de atualização de topologia.

informações são organizadas em triplas contendo uma aresta, a situação (funcional ou falho) da aresta e um contador para ordenação das situações (timestamp).

Quando um nó t recebe uma mensagem de atualização, ele chama o procedimento `ReceiveUpdateMessage(msg, t)`, descrito na figura 3.5. Esse procedimento é responsável tanto por atualizar a topologia do nó que recebe a mensagem, quanto por adicionar as informações de alterações nas informações a serem enviadas pelo procedimento `Topology Maintainer(s)`.

Ao receber uma confirmação de recebimento de mensagem de atualização, o nó executa o procedimento `ReceiveUpdateAckMessage(msg, t, l)`, descrito na figura 3.6. Esse procedimento remove as informações entregues das informações a serem enviadas pelo nó.

Caso não seja recebida nenhuma mensagem de um determinado nó após um *time-out*, parametrizável, a aresta correspondente é considerada falha. Este *time-out* é chamado de β, sendo que $\beta > \alpha$.

Uma vez conhecido o processo de manutenção da topologia pode-se especificar o procedimento responsável por enviar uma mensagem de uma origem até o seu destino, conforme mostra a figura 3.7. Suponha que s e t são os nós que correspondem à origem e ao destino desse processo de roteamento, como exemplificado na figura 3.1. Quando o nó s deseja enviar uma mensagem ao nó t, assume-se que, é chamado o procedimento `SendMessage(payload, s, t, needsAck)`, passando o conteúdo da mensagem (*payload*), a origem (s), o destino (t) e a indicação da necessidade de envido de uma mensagem de confirmação por parte do destino (*needsAck*).

No recebimento de uma mensagem por um nó, u, supõe-se que, é chamado o procedimento `ReceiveMessage(msg, u, l)`, que é responsável por: executar os procedimentos `ReceiveUpdateMessage(msg, u)` e `ReceiveUpdateAckMessage(msg, u, l)`, caso seja recebida uma mensagem de atualização de topologia; entregar o conteúdo da mensagem ao nível de aplicação, caso seja recebida uma mensagem cujo destino é o próprio nó; ou executar o procedimento de roteamento (`RouteMessage(msg, node)`), caso não seja o destino final da mensagem.

O procedimento `RouteMessage(msg, node)`, descrito na figura 3.9, é utilizado como

```
ReceiveUpdateMessage(msg, t)
 1. Let s ← msg.Source
 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.LasEdgeInformation[i.Edge]
17.         If i' does not exist 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 3.5: 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 3.6: Especificação do algoritmo executado no recebimento da confirmação de uma mensagem de atualização.

```
SendMessage(payload, s, t, needsAck)
```

1. Let $msg.Payload \leftarrow payload$
2. Let $msg.Src \leftarrow s$
3. Let $msg.Dest \leftarrow t$
4. Let $msg.NeedsAck \leftarrow needsAck$
5. Let $msg.Visited \leftarrow anEmptySet$
6. Run `RouteMessage(msg, s)`

Figura 3.7: Especificação do algoritmo de roteamento: SendMessage.

```
ReceiveMessage(msg, u, l)
```

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

Figura 3.8: Especificação do algoritmo de roteamento: ReceiveMessage.

```
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 $CandLinks \leftarrow$ 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 $WorkingGraph \leftarrow node.KnownTopology$
12. Remove all nodes in $msg.Visited$ from $WorkingGraph$
13. If $CandLinks$ contains at least one link Then
14. Let $l \leftarrow$ the link in $CandLinks$ with largest $\Gamma(WorkingGraph, l)$
15. Send message msg through l
16. Else
17. Remove last node from $msg.Visited$
18. If $msg.Visited$ contains at least one node Then
19. Let $l \leftarrow$ the previous node in $msg.Visited$
20. Send an update message to l
21. Send msg back to l
22. Else
23. Return Error: There is no route to $msg.Dest$

Figura 3.9: Especificação do algoritmo de roteamento: RouteMessage.

procedimento auxiliar dos demais procedimentos. Sua função é escolher a aresta apropriada e enviar a mensagem por essa aresta em direção ao destino final da mensagem. A função $\Gamma(WorkingGraph, l)$, que é discutida em maior detalhes na seção 3.2, efetua o cálculo do compromisso, *trade-off*, para a seleção da aresta para a rota.

Na seleção das arestas a serem utilizadas no procedimento `RouteMessage(msg, node)` são desconsideradas as arestas que levam a nós já visitados pela mensagem (nós em $msg.Visited$). Esse tratamento é feito para evitar ciclos no caminho percorrido.

Há, porém o caso específico em que arestas que geram ciclos são consideradas. Esse caso pode ocorrer, por exemplo, caso as informações da topologia em dois nós distintos da rede estejam desatualizadas, isto é, um nó pode ter sido escolhido para o roteamento em virtude de caminhos que não existem mais, ou que estejam falhos, e no momento em que a mensagem alcança esse nó, o qual possui informações mais atualizadas, é verificado que as únicas arestas que possuem caminhos até o destino levam a nós já previamente visitados.

Nesse caso, é necessário que a mensagem seja retornada pela aresta a partir da qual chegou ao nó. Porém, o nó que receber essa mensagem, provavelmente, terá as informações desatualizadas sobre a rede, visto que o selecionou para o roteamento um nó sem opções de rotas. Logo, para que esse nó possua as informações atuais, a mensagem de roteamento programada para ser enviada para esse nó é antecipada, sendo enviada anteriormente à mensagem que contém o *payload*. Desta forma, o nó pode tomar a decisão de um novo caminho baseando-se nas informações atualizadas da topologia.

3.3.2 Exemplificação do Funcionamento do Algoritmo

A figura 3.10 demonstra o funcionamento da execução do algoritmo na rede apresentada anteriormente. Inicialmente, considerando que o nó s precisa enviar uma mensagem para o nó t, o algoritmo é executado em s, e avalia suas arestas adjacentes com o objetivo de qual aresta será utilizada para roteamento, ou seja, as arestas (s, a) e (s, f) são avaliadas.

Caso a aresta (s, a) seja escolhida (o que de fato ocorre, pois o cálculo de *trade-off* considera, também, o número de desvios que são possíveis a partir de cada nó avaliado),

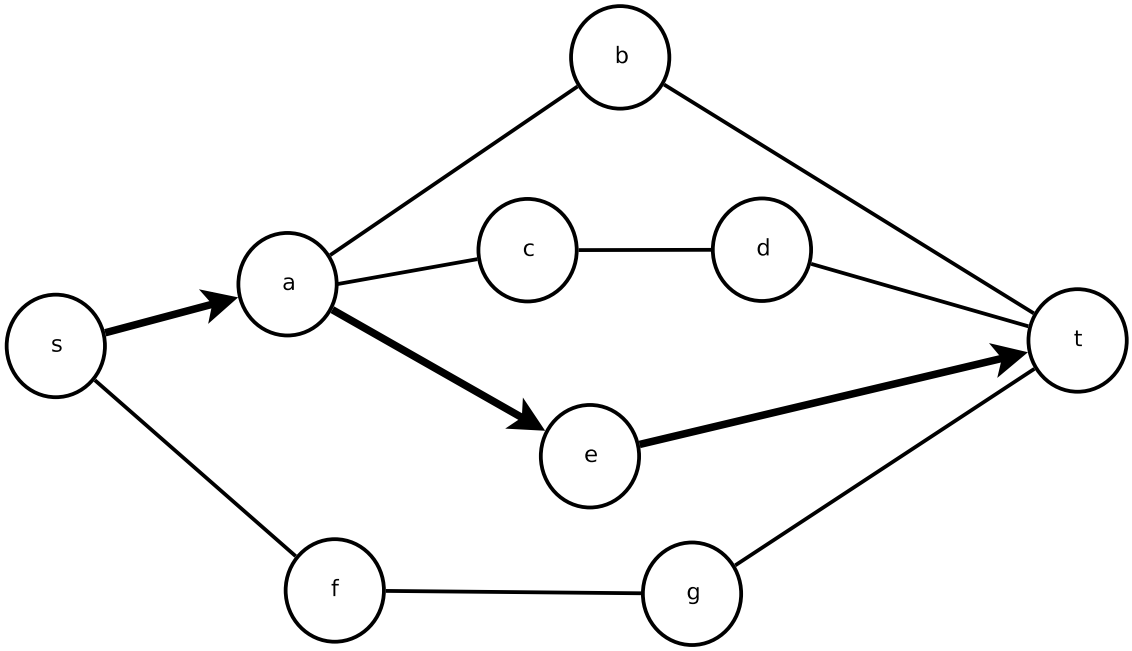

Figura 3.10: A representação do roteamento em uma rede.

a mensagem é enviada para a. Quando o nó a recebe a mensagem com destino t é executada a função de roteamento, avaliando as arestas $(a, b), (a, c)$ e (a, e). A aresta (a, s) é descartada da avaliação, visto que ela leva ao nó s, que já foi visitado.

Nesse passo, as arestas que apresentam um maior valor para o cálculo de compromisso são (a, b) e (a, e), pois apresentam um caminho menor do que seria possível por (a, c). Suponha que seja escolhida a aresta (a, e); de forma semelhante o nó e executa o algoritmo de roteamento e, percebendo que possui uma aresta o que conecta diretamente ao nó de destino da mensagem t, envia a mensagem pela aresta (e, t) e finalizando o processo de roteamento para essa mensagem.

Suponha que, na mesma rede, o nó s precise enviar outra mensagem a t. Porém, dessa vez, a aresta (e, t) torna-se indisponível. A figura 3.11 representa o funcionamento do algoritmo no caso da ocorrência dessa falha durante o roteamento da mensagem. Considerando que a falha tenha ocorrido logo após o inicio do processo de roteamento da mensagem de s para t, apenas os nós e e t, vizinhos da aresta falha, possuem uma representação atualizada da topologia da rede.

O procedimento de roteamento, inicialmente, ocorre de maneira idêntica à anterior, visto que as informações sobre a topologia da rede não foram atualizadas nos nós ditantes

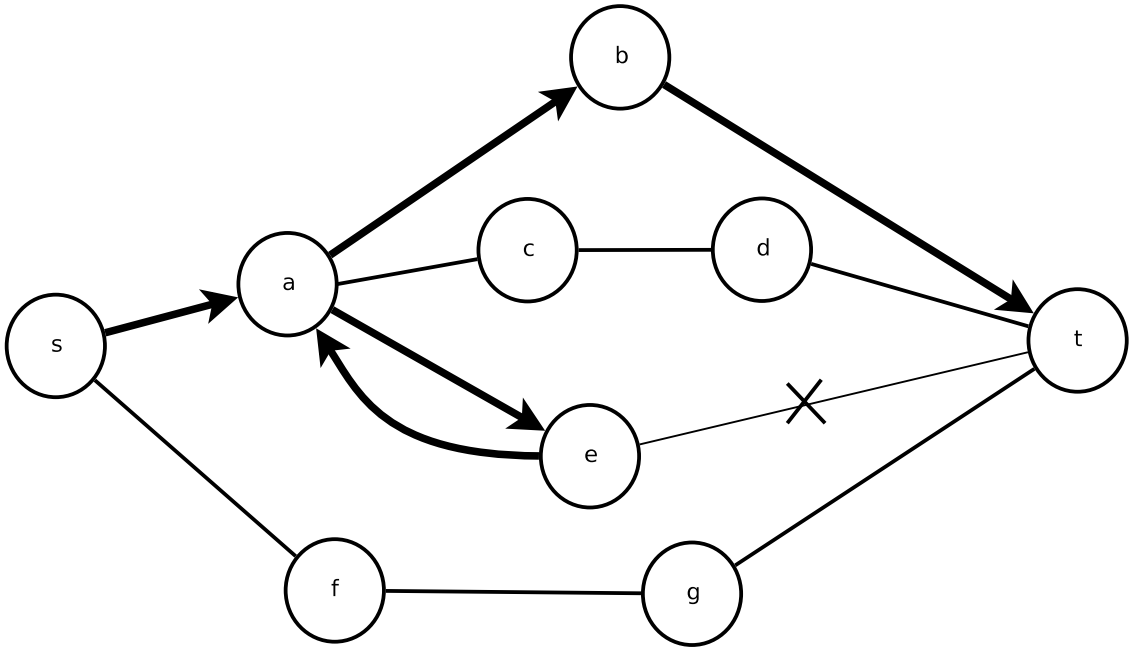

Figura 3.11: A representação do roteamento em uma rede na presença de uma falha

da falha. Ou seja, o algoritmo é executado no nó s, enviando a mensagem ao nó a, que por sua vez a envia ao nó e.

Quando o nó e receber a mensagem destinada a t, será verificado que os únicos caminhos possíveis para o destino t são através do nós já visitados pela mensagem. Como não há um caminho para o destino que utilize nós que não tenham sido visitados pela mensagem, o nó e se prepara para enviar a mensagem ao nó a, que é o nó imediatamente anterior no caminho.

Para evitar a ocorrência de um ciclo, e antecipa o envio da mensagem de atualização da topologia para a, que seria enviada nos próximos α segundos pelo processo `TopologyMaintainer`, descrito na figura 3.4. Com o recebimento dessa mensagem, a atualiza a sua representação da topologia e pode decidir corretamente pelo caminho a ser utilizado para o roteamento. Observe que, se a mensagem de atualização de topologia não fosse enviada, o nó a continuaria tomando a decisão de enviar a mensagem ao nó e baseado nas informações desatualizadas da topologia.

Após o envio da mensagem de atualização por e, é retornada a mensagem contendo o *payload* para o nó a que, mediante a nova informação obtida, decide enviar a mensagem para o nó b. Esse nó envia a mensagem para o nó t através da aresta (b, t), finalizando o

processo de roteamento.

3.4 Avaliação da Complexidade do Algoritmo

O algoritmo descrito nesse capítulo baseia-se no cálculo do fluxo máximo efetivo entre os nós avaliados para o roteamento e o destino de uma mensagem. Para calcular o fluxo máximo efetivo são desconsiderados os caminhos disjuntos que utilizam nós já visitados pela mensagem roteada. Dessa forma, esse processo de roteamento é dependente do contexto de *cada* mensagem, mais especificamente, do conjunto de nós visitados por cada mensagem a cada nó percorrido. Portanto, no decorrer dessa seção, é apresentada uma avaliação desse algoritmo considerando sua complexidade de roteamento por mensagem.

Para calcular a complexidade de roteamento de uma mensagem, precisa-se calcular a complexidade da seleção de uma aresta para o encaminhamento dessa mensagem e o número de nós que essa mensagem percorre até chegar ao seu destino. Por sua vez, a complexidade da seleção de uma aresta depende da complexidade da função de avaliação e do número de arestas avaliadas. Nos próximos parágrafos é analisada a complexidade de cada um desses passos.

A complexidade da função de avaliação de uma aresta e, $(\Gamma(G, e))$, considerando apenas os dois critérios citados na seção 3.2, é a soma da complexidade do cálculo do fluxo máximo, $O(V * E)$, com a complexidade para o cálculo de comprimento do caminho mínimo, $O(E)$, produzindo a complexidade $O(V * E + E)$, ou simplesmente $O(V * E)$ [50].

Como a função de avaliação é executada para cada uma das arestas vizinhas a um nó v, a complexidade de seleção de uma aresta é a complexidade da função de avaliação $\Gamma(G, e)$ multiplicada pelo número de arestas adjacentes ao nó v (g_v), resultando em $O(V * E * g_v)$. Como $N * g_v = 2 * E$, *i.e.*, o número de nós vizinhos em um grafo é o dobro do número de arestas, a seleção de uma aresta é efetuada com custo $O(E^2)$.

De acordo com o algoritmo Schroeder-Duarte, no envio de uma mensagem, cada um dos nós pode, no pior caso, ter essa mensagem retornada por todas as suas arestas adjacentes. Ou seja, uma mensagem pode percorrer todas as arestas de um grafo sendo retornada após tentativas mal sucedidas de roteamento. Dessa forma, o número de nós

visitados por uma mensagem é proporcional ao número de arestas E do grafo, ou seja, $O(E)$.

Portanto, a complexidade de roteamento de uma mensagem, no pior caso, é $O(E^2 * E)$ (a complexidade de seleção de uma aresta multiplicada pelo número de nós visitados), ou seja, $O(E^3)$. Em outras palavras, para rotear cada mensagem, o algoritmo descrito nesse capítulo efetua um processamento com custo $O(E^3)$, sendo o custo de $O(E^2)$ para a escolha de cada aresta da rota.

Uma complexidade $O(E^3)$ por mensagem roteada não propicia uma implementação prática desse algoritmo para redes de alta velocidade. Sendo assim, este trabalho visa tornar prático o roteamento robusto baseado em avaliação do fluxo máximo através do algoritmo apresentado no capítulo seguinte.

Capítulo 4

O Algoritmo Heurístico de Roteamento Robusto

Este capítulo apresenta o algoritmo heurístico proposto, o qual viabiliza, na prática, o roteamento robusto baseado em avaliação do fluxo máximo. A heurística utilizada por esse algoritmo tem a finalidade de permitir a geração de estimativas da robustez dos caminhos da rede. Essas estimativas são baseadas nos critérios definidos no capítulo 3 e armazenadas em tabelas de roteamento. Essas tabelas são atualizadas periodicamente, conforme alterações na topologia da rede, e são utilizadas pelo processo de encaminhamento de mensagens.

A utilização de tabelas de roteamento contrasta com o algoritmo do capítulo anterior, que realiza um cálculo completo das estimativas sempre que uma mensagem é encaminhada. Esse cálculo é inerente à utilização do contexto das mensagens na avaliação das arestas candidatas ao roteamento, o que implica na não utilização de tabelas de roteamento. Pois, uma tabela que contemplasse todas as situações possíveis para o encaminhamento de mensagens teria um número de entradas proporcional a $O(V!)$, com V sendo o número de nós da rede. Portanto, impraticável tanto pela memória, quanto pelo processamento necessários para a criação e manutenção da tabela.

Como o algoritmo do capítulo anterior apresenta uma complexidade $O(E^2)$ por mensagem encaminhada e $O(E^3)$ por mensagem roteada, não é factível uma implementação

prática desse algoritmo para redes de alta velocidade. Para resolver esse problema, o algoritmo proposto neste trabalho efetua o encaminhamento de mensagens individuais apenas consultando a tabela de roteamento. Neste sentido, a complexidade do algoritmo é transferida para a construção e manutenção das tabelas.

A especificação do algoritmo heurístico proposto é apresentada a seguir. Logo após, na seção 4.2, é descrito e exemplificado o funcionamento desse algoritmo. Por fim, a seção 4.3 apresenta uma prova formal de sua correção.

4.1 Especificação do Algoritmo Proposto

O algoritmo proposto baseia-se em representações da topologia mantidas individualmente por cada nó pertencente à rede, da mesma forma que o algoritmo apresentado no capítulo 3. Como o algoritmo proposto também escolhe apenas a próxima aresta do caminho até o destino a cada nó, não é necessário que a representação da topologia mantida pelo nó que executa o algoritmo reflita todas as alterações ocorridas na rede.

Para a manutenção da topologia pelos nós da rede são executados os procedimentos de atualização apresentados no capítulo 3 com algumas modificações. Essas modificações proporcionam a geração das tabelas de roteamento utilizadas para a seleção das arestas durante o processo de roteamento.

Conforme ocorrem alterações na topologia da rede, os nós atualizam suas representações da topologia mediante o recebimento de mensagens de atualização ou mediante a detecção de alterações em arestas adjacentes. Sempre que uma alteração é efetuada na representação da topologia, a tabela de roteamento do nó é recalculada.

O procedimento `TopologyMaintainer(s)` apresentado na figura 4.1 é iniciado por cada nó s no momento em que entra na rede e, a cada α segundos avalia a situação das arestas mantidas na estrutura $KnownTopology$, que contém apenas arestas sem falha. Ao detectar uma falha, utilizando *timers*, o nó envia mensagens de atualização aos nós vizinhos. O estado das arestas em $KnownTopology$ é reportado em mensagens que são enviadas mesmo que não ocorram alterações na topologia, servindo para zerar os *timers* utilizados no teste das arestas. Caso não sejam recebidas mensagens de um nó vizinho,

```
TopologyMaintainer(s)
```

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

Figura 4.1: Especificação do processo de atualização de topologia.

o *timer* correspondente ultrapassa o valor limite e a aresta que liga a esse nó é marcada como inválida. Também durante cada intervalo de α segundos, caso ocorram alterações na topologia, é recalculada a tabela de roteamento.

A mensagem de atualização de topologia enviada pelo nó s é recebida pelo nó t. Na ocasião de recebimento de uma mensagem de atualização de topologia, o nó t executa o procedimento `ReceiveUpdateMessage(msg, t)`, descrito na figura 4.2. Esse procedimento é responsável tanto por atualizar a topologia do nó que recebe a mensagem, quanto por adicionar as informações sobre alterações nas mensagens a serem enviadas pelo procedimento `TopologyMaintainer`. Ao receber uma mensagem de atualização, se a aresta pela qual a mensagem foi recebida não está em $KnownTopology$, ela é incluída. O nó que recebe a mensagem deve verificar, em seu conteúdo, todas as informações recebidas sobre a alteração da topologia. Estas informações devem ser adequadamente enviadas aos demais vizinhos.

Ao receber uma mensagem de atualização, é enviada uma mensagem de confirmação de recebimento. Ao receber essa confirmação é executado o procedimento `ReceiveUpdate AckMessage(msg,t,l)`, apresentado pela figura 4.3. Esse procedimento faz o nó t remover as informações confirmadas da lista $EdgeInformationTo\ Send$.

O procedimento usado para calcular a tabela de roteamento, `GenerateRoutingTable(s)`, é descrito na figura 4.4. Esse procedimento é executado por um nó s sempre que ocorrer uma atualização na sua representação da topologia, e gera como resultado uma tabela de roteamento contendo todos os destinos possíveis. Porém, para cada entrada, ou destino, da tabela de roteamento é mantida uma *lista de arestas candidatas ao roteamento* que marca entre as arestas adjacentes as que possuem um caminho até o destino, ordenadas conforme a função de avaliação $\Gamma(WorkingGraph, e)$, apresentada na seção 3.2.

Para a avaliação das arestas adjacentes, durante o processo de geração da tabela de roteamento, *o nodo que executa o processo de roteamento remove a si próprio de sua representação interna da topologia*. Dessa forma, é medida apenas uma estimativa do fluxo máximo para a avaliação. Não se pode remover os nós, ou arestas, que tenham sido visitados pelas mensagens, uma vez que a mesma tabela é usada para rotear mensagens de

```
ReceiveUpdateMessage(msg, t)
 1. Let s ← msg.Source
 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 link l' adjacent to t
12.                 Add i to t.EdgeInformationToSend[l']
13.         For each link information i' available in t.LastEdgeInformation
14.                 Add i' to t.EdgeInformationToSend[l]
15. For each link information i in msg.Payload
16.         Let i' ← t.LastEdgeInformation[i.Edge]
17.         If i' does not exist OR i'.TimeStamp < i.TimeStamp
18.                 Let t.LastEdgeInformation[i.Edge] ← i
19.                 For each link l' adjacent to t
20.                         Add i to t.EdgeInformationToSend[l']
21.                 If i.Working = TRUE
22.                         Add link i.Edge to t.KnownTopology
23.                 Else
24.                         Remove link i.Edge from t.KnownTopology
```

Figura 4.2: Especificação do algoritmo executado no recebimento de uma mensagem de atualização.

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

Figura 4.3: Especificação do algoritmo executado no recebimento da confirmação de uma mensagem de atualização.

```
GenerateRoutingTable(s)
1. Let WorkingGraph ← s.KnownTopology − {s}
2. For each destination node t in WorkingGraph
3.         Let CandidateLinks ← an empty list to possible links out
4.         For each link l adjacent to s in s.KnownTopology
5.                 If l does not have a path to t
6.                         Ignore l
7.                 Else
8.                         Add l to CandidateLinks
9.         If CandidateLinks contains more than one link
10.                Sort links in CandidateLinks according to the evaluation function
   Γ(WorkingGraph, l′)
11.        Let RoutingTable[t] ← CandidateLinks
```

Figura 4.4: Especificação do algoritmo de roteamento: geração da tabela de roteamento.

qualquer origem para qualquer destino. Ou seja, no algoritmo proposto, o fluxo máximo é avaliado de uma forma independente do contexto das mensagens.

A partir da tabela de roteamento produzida, é possível definir os procedimentos para envio e recebimento de mensagens como segue. Para o envio de uma mensagem entre um nó de origem s e um nó de destino t, o nó s executa o procedimento `SendMessage (payload, s, t, needsAck)`, apresentado na figura 4.5. Esse procedimento recebe como entrada o conteúdo da mensagem (*payload*), o nó de origem (s), o nó de destino (t) e a indicação da necessidade de envio de uma mensagem de confirmação de recebimento por parte do destino (*needsAck*).

O procedimento `PacketForwarding(msg, node)`, descrito na figura 4.7, é utilizado como auxiliar pelos procedimentos de envio e recebimento de mensagem. Ele é responsável por escolher a aresta apropriada, consultando a tabela de roteamento, e enviar a mensagem por essa aresta em direção ao seu destino final.

Ao receber uma mensagem é executado o procedimento `ReceiveMessage(msg, u, l)`, apresentado na figura 4.6. Esse procedimento é responsável por efetuar os seguintes passos: encaminhar a mensagem recebida ao nível de aplicação, caso seja o destino final da mensagem; encaminhar a mensagem ao procedimento de recebimento de mensagem de

```
SendMessage(payload, s, t, needsAck)
```

1. Let $msg.Payload \leftarrow payload$
2. Let $msg.Src \leftarrow s$
3. Let $msg.Dest \leftarrow t$
4. Let $msg.NeedsAck \leftarrow needsAck$
5. Let $msg.Visited \leftarrow$ um conjunto vazio
6. Run `PacketForwarding(msg, s)`

Figura 4.5: Especificação do algoritmo de roteamento: envio de mensagem.

```
ReceiveMessage(msg, u, l)
```

1. If $msg.Dest = u$
2. If msg is an update message
3. Run `ReceiveUpdateMessage(msg, u)`
4. Else If msg is ACK to an update message
5. Run `ReceiveUpdateAckMessage(msg, u, l)`
6. Else
7. Send $msg.Payload$ to the Application-Level
8. If $msg.NeedsAck$
9. Run `SendMessage(Ack(msg), u, msg.Src, FALSE)`
10. Else
11. Run `PacketForwarding(msg, u)`

Figura 4.6: Especificação do algoritmo de roteamento: recebimento de mensagem.

atualização, caso seja uma mensagem de atualização de topologia; enviar uma mensagem de confirmação de recebimento caso a mensagem recebida necessite de tal confirmação; encaminhar a mensagem ao procedimento de recebimento de confirmação de mensagens, caso seja uma mensagem de confirmação; e, finalmente, por encaminhar a mensagem ao procedimento `PacketForwarding(msg, u)`, caso a mensagem não tenha atingido seu destino final.

O procedimento de encaminhamento (*Forwarding*) efetua primeiramente uma verificação para detectar se a mensagem a ser roteada não está contida em um ciclo. Os ciclos são permitidos pelo algoritmo proposto quando o nó que efetua o roteamento não possui uma alternativa de rota, na qual o próximo nó não esteja na lista de nós visitados pela mensagem, até o destino final da mensagem. Caso seja necessária a criação de um ciclo, a mensagem é retornada ao nó imediatamente anterior no caminho percorrido.

Caso a mensagem não esteja contida em um ciclo, é efetuada uma busca na tabela de roteamento pela aresta que possui maior resultado de avaliação e que ainda não tenha sido visitada. Se existir uma aresta que atenda a essas condições a mensagem é roteada para essa aresta, caso contrário é gerado um ciclo. Se o nó for o inicial, e não há alternativa de rota, é gerada uma mensagem de erro informando à aplicação que a mensagem não pode ser roteada.

Na ocorrência de um ciclo, são realizados os mesmos passos descritos acima, mas como o nó que gera o ciclo está marcado como visitado ele não será escolhido novamente para o roteamento. Ou seja, o ciclos apenas retrocedem ao nó imediatamente anterior para a tentativa de outros caminhos. Caso não seja encontrado nenhum caminho até o destino, a mensagem é retornada nó a nó, até que seja gerado um erro de falta de rota.

Para realizar esse processo são necessárias duas estruturas, uma para armazenar apenas o caminho principal percorrido pelas mensagens, e outra para gravar os nós visitados. Quando um nó gera um ciclo, ele deixa de pertencer ao caminho principal, sendo considerado apenas como um nó visitado. Com isso, o nó gerador do ciclo é escolhido para roteamento novamente, e caso seja necessário que a mensagem retroceda mais um nó, esta será retrocedida para o nó imediatamente anterior no caminho principal de roteamento.

```
PacketForwarding(msg, node)
```

1. If $msg.Dest$ is adjacent to $node$
2. Let $l \leftarrow$ the link that goes directly from node $node$ to $msg.Dest$
3. Send message msg through link l
4. Else If $msg.Visited$ does not contain the node $node$
5. Add node $node$ to $msg.Visited$
6. If $RoutingTable[msg.Dest]$ contains nodes that are not in $msg.Visited$
7. Let $l \leftarrow$ the link with largest $RoutingTable[msg.Dest]$ whose destination is not in $msg.Visited$
8. Send message msg through l
9. Else // Outdated topologies OR it is not a connected graph
10. If $msg.Visited$ contains at least one node before $node$
11. Let $l \leftarrow$ the link to the node before $node$ in $msg.Visited$
12. Send msg back to l
13. Else
14. Return Error: There is no route to $msg.Dest$
15. Else // A cycle was detected
16. If $RoutingTable[msg.Dest]$ contains nodes that are not in $msg.Visited$
17. Let $l \leftarrow$ the link with largest $RoutingTable[msg.Dest]$ whose destination is not in $msg.Visited$
18. Send message msg through l
19. Else // Outdated topologies OR it is not a connected graph
20. If $msg.Visited$ contains at least one node before $node$
21. Let $l \leftarrow$ the link to the node before $node$ in $msg.Visited$
22. Send msg back to l
23. Else
24. Return Error: There is no route to $msg.Dest$

Figura 4.7: Especificação do algoritmo de roteamento: encaminhamento de mensagem.

4.2 Exemplificação do Algoritmo Proposto

As figuras 4.8 e 4.9 apresentam exemplos de roteamento utilizando o algoritmo proposto em topologias simples. A figura 4.8 apresenta uma situação de falha no enlace (b, t), o qual seria utilizado para o roteamento da mensagem. No momento em que a mensagem considerada é enviada pelo nó s, apenas o nó b tem o conhecimento da falha. Dessa forma, inicialmente o nó s envia a mensagem para o nó a, pois é sua única alternativa de roteamento. Considerando que a tabela de roteamento apresenta o nó b como a alternativa que possui o maior valor para a função de avaliação, a mensagem é então roteada ao nó b. Porém, como o nó b não possui alternativas de roteamento que não utilizem nós já visitados pela mensagem, a mensagem é então retornada ao nó imediatamente anterior de sua rota percorrida, a.

Ao receber a mensagem novamente, a entrada da tabela que possui maior valor para função de avaliação, e que não tenha sido visitada previamente pela mensagem roteada é a entrada que indica o nó c como destino. Então, o nó a roteia a mensagem ao nó c. Nesse ponto o caminho principal percorrido pela mensagem é representado como $s- > a- > c$, sendo também o nó b marcado como visitado, porém não pertencente ao caminho principal.

Considerando que a tabela de roteamento do nó c apresente para o destino t as alternativas a, h, d, ordenadas segundo a função de avaliação Γ, o nó c irá, inicialmente optar por a. Porém como o nó a já está marcado como visitado na mensagem roteada é então escolhido o nó h para o roteamento da mensagem.

Vale destacar que a avaliação do nó h apresenta um valor maior que a do nó d. Pois, quando é desconsiderado o nó c e todas as suas arestas adjacentes para o cálculo da tabela em c, apesar da avaliação dos nós h e d possuírem o mesmo valor para o critério de fluxo máximo (um), o caminho mínimo para o nó t é menor por h ($h- > a- > b- > t$), dado que c desconhece a falha da aresta (b, t).

Da mesma forma que em b, ao chegar no nó h, como não existem alternativas para o roteamento que não tenham sido visitadas pela mensagem, esta é retornada ao nó imediatamente anterior da rota percorrida, c. Então, é considerado o nó d para o roteamento, pois é o único que não foi percorrido pela mensagem. A partir do nó d, é percorrido o

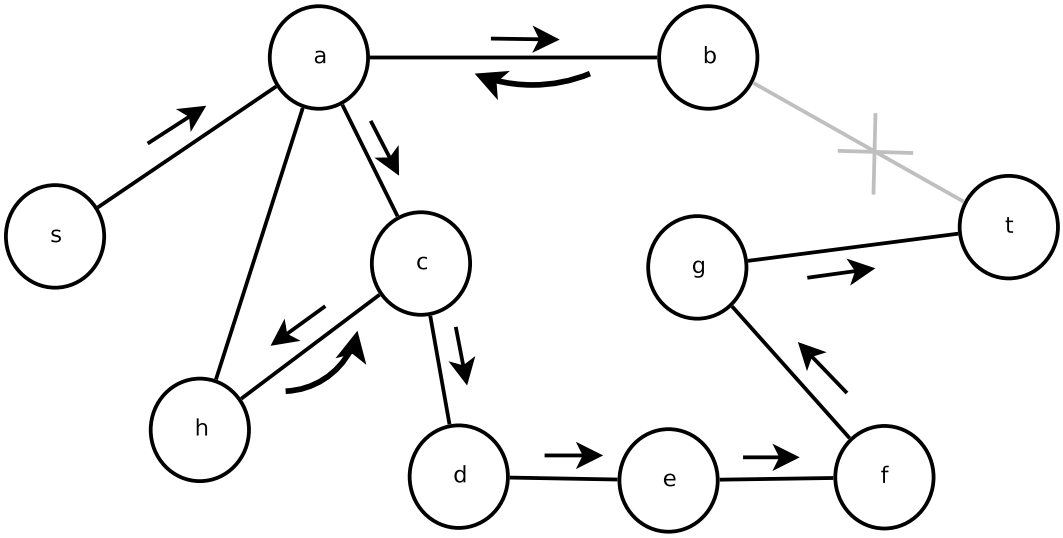

Figura 4.8: Uma rede com falha em uma de suas arestas.

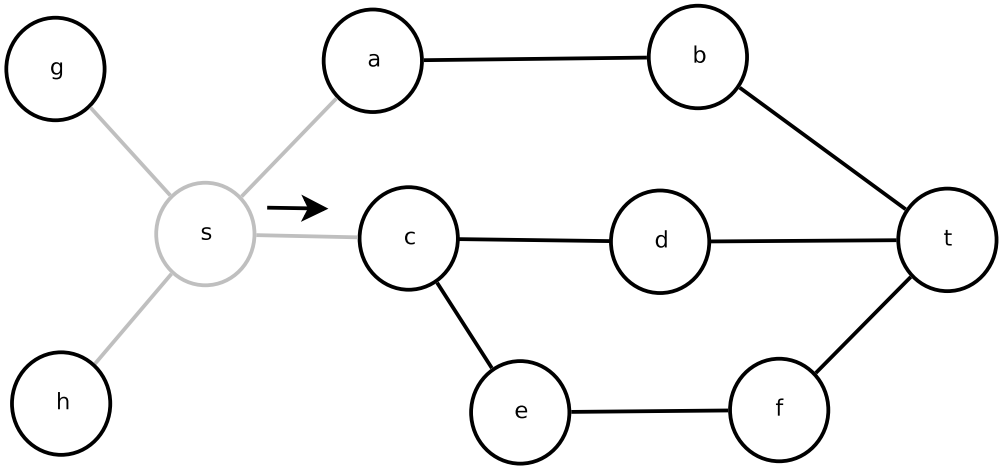

Figura 4.9: A escolha de uma aresta em um ponto de corte.

caminho $e->f->g->t$ até o destino final da mensagem.

Nesse exemplo é possível observar a diferença desta proposta para o algoritmo do capítulo 3, que está na escolha do nó h. Pois, seriam desconsiderados os nós s, a, b para o roteamento da mensagem no nó c, e a mensagem seria roteada diretamente ao nó d. Entretanto, apesar do nó h ter sido escolhido para o roteamento, a mensagem é roteada com sucesso até seu destino final, e esse roteamento é feito apenas consultando a tabela de roteamento de cada nó, sem a necessidade de recalcular o próximo nó da rota a cada aresta percorrida. A utilização desse formato para a tabela de roteamento apresenta várias situações satisfatórias ao roteamento, duas delas são apresentadas na figura 4.9, e descritas a seguir.

Para o exemplo apresentado na figura 4.9, que representa uma situação de escolha de aresta em ponto de corte, ao calcular-se a tabela de roteamento do nó s, são considerados os nós a, c, g e h. Porém, como o nó s e todas suas arestas são desconsideradas durante o cálculo, os únicos nós que possibilitam o roteamento de mensagens ao nó t são os nós a e c e, portanto, são as únicas entradas inseridas na tabela de s para o destino t.

Como a função de avaliação retorna um valor melhor para o nó c, pois através desse nó é possível rotear uma mensagem ao nó t através de dois caminhos disjuntos, a aresta (s, c) é escolhida preferencialmente à aresta (s, a). Desse modo, a mensagem é roteada ao nó c. Na ocorrência de falhas na rede é clara a vantagem dessa escolha, pois, caso seja necessária a utilização de um desvio, a escolha do nó c aumenta a probabilidade desse desvio ser menor.

Nesse caso, como a rede não apresenta falhas, a mensagem é roteada através do caminho $c- > d- > t$ até o seu destino final. Esse caminho, além de ser um dos caminhos mínimos possíveis, é o que apresenta maior probabilidade de utilização de um desvio mais curto, na ocorrência de falhas na rede.

4.3 Prova de Correção do Algoritmo

Para provar a correção do algoritmo é formulado o Teorema 4.1 que afirma que, 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 a sua correção.

A primeira hipótese tem relação com o conhecimento por parte dos nós do estado de suas arestas e nós adjacentes. Não existe diferenciação para um nó da rede se a falha está na aresta ou no nó adjacente.

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.

Essa segunda hipótese é suficiente, porém não é necessária para o funcionamento do algoritmo. Uma mensagem pode ser roteada mesmo que o nó de origem não conheça nenhum caminho válido até o destino. Caso o nó de origem encaminhe esta mensagem a qualquer nó que conheça um caminho caminho não falho até o nó de destino, ela será roteada corretamente.

Na terceira hipótese assume-se que não há 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. Essa hipótese também garante que as tabelas de roteamento estejam calculadas em cada um dos nós da rede antes que uma mensagem seja enviada.

Para a prova de correção do algoritmo, inicialmente é provado um lema que indica que, se o algoritmo funciona para uma seqüência de ordenação das arestas candidatas ao roteamento na tabelas de cada nó, e se for alterada esse ordenação o algoritmo continuará funcionando. O lema é provado por indução. A seguir, é provado, também por indução, o teorema de correção do algoritmo.

Lema 4.1. Considere $G = (V, E)$ um grafo, e $s, t \in V$ dois nós não falhos desse grafo. Considere que para uma certa ordenação das arestas candidatas, listadas na entrada correspondente ao destino t das tabelas de roteamento de cada nó, o algoritmo proposto nesse trabalho funciona corretamente. Se for alterada a ordenação das arestas candidatas ao roteamento nas tabelas dos nós do grafo G, e se s enviar uma mensagem para t utilizando o algoritmo proposto nesse trabalho, a mensagem chegará até t.

Prova. Esta prova considera, sem perda de generalidade, apenas um nó x pertencente a um caminho válido da origem ao destino, com exceção do nó de destino da mensagem.

Essa consideração é feita sem perda de generalidade, pois, se em cada nó existe uma lista de arestas e uma que leva a um caminho válido é selecionada, o destino final é atingido em algum momento. Assume-se que a tabela de roteamento de x, para a entrada de destino t, apresente o seguinte conjunto de arestas candidatas ao roteamento $e_1, e_2, ..., e_{n-1}$. Existe ao menos uma aresta nesse conjunto que leva a um nó pertencente a um caminho válido para o qual a mensagem deve ser enviada.

Como base de uma indução, é assumido que o conjunto de arestas candidatas contém apenas um elemento. Neste caso, a prova de que a alteração da ordem de escolha das arestas candidatas não afeta o roteamento é trivial, pois o conjunto continua ordenado da mesma maneira.

Como desenvolvimento da indução, é assumida a hipótese de que o lema é verdadeiro para o conjunto de arestas $e_1, e_2, ..., e_{n-1}$. Deve-se provar que o mesmo lema é verdadeiro para o conjunto de arestas $e_1, e_2, ..., e_n$.

Assume-se, novamente sem perda de generalidade, que a aresta escolhida para o roteamento é a aresta e_n. Portanto, a mensagem será enviada para o nó que está conectado à outra ponta dessa aresta, chamado de u. Se o nó u fizer parte de um caminho válido para o destino t está provado que o lema é válido para o novo conjunto de arestas. Caso contrário, após o nó u tentar enviar a mensagem para todas as alternativas possíveis de roteamento que não tenham passado por nós visitados pela mensagem, a mensagem é retornada ao nó x e a aresta e_n, conforme o algoritmo, não será utilizada novamente para o roteamento.

Portanto, o conjunto de arestas disponíveis será $e_1, e_2, ..., e_{n-1}$, para o qual o lema é verdadeiro. Logo, prova-se que que o algoritmo continua funcionando caso seja alterada a ordenação das arestas candidatas nas tabelas de roteamento.

Teorema 4.1. Considere $G = (V, E)$ um grafo, e $s, t \in V$ dois nós não falhos desse grafo. Considere que os nós possuem conhecimento do estado 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 base para uma indução, assume-se que o grafo G possui dois nós. Esses nós correspondem aos nós s e t que, por definição são diferentes. A única aresta candidata ao roteamento presente na tabela de roteamento do nó s é a aresta (s, t), visto que não existem outras possibilidades e, pela hipótese, existe no mínimo um caminho para o destino t que esteja funcional e seja conhecido por s. Como a aresta (s, t) é funcional, a mensagem será enviada pelo nó s através dessa aresta, chegando ao seu destino t e confirmando a base da indução.

Assumindo como hipótese para a indução que o teorema é válido para um grafo com $(n-1)$ nós, deve-se provar que o teorema também é válido para um grafo com n nós. Considerando, então, que o grafo H possui $(n-1)$ nós e o teorema é válido para esse grafo conforme a hipótese anterior. Suponha que o grafo I seja um grafo resultante da inclusão de um nó n e da inclusão de até $(n-1)$ arestas, conectadas ao nó n, no grafo H.

Considerando, sem perda de generalidade, dois nós s e t comuns aos grafos H e I, suponha que o nó s envia uma mensagem destinada ao nó t. Nessa situação, existem dois casos iniciais $CASO1, CASO2$ que precisam ser provados para aceitar o teorema como válido para qualquer número de nós. No primeiro, a inclusão do nó n e das arestas que conectam esse nó aos demais nós do grafo I não interfere, mesmo considerando possíveis alterações nas tabelas de roteamento geradas nos nós, no envio da mensagem do nó s ao nó t. Nesse caso o teorema é válido de acordo com a hipótese.

No segundo caso possível ($CASO2$), a inclusão do nó n, e suas arestas adjacentes, interfere no envio da mensagem do nó s ao nó t. Vale ressaltar que essa inclusão pode apenas alterar a ordenação das arestas candidatas nas tabelas de roteamento e/ou incluir uma aresta que aponte para o nó n. Nesse caso, existem algumas possibilidades listadas a seguir.

A primeira possibilidade ($CASO2.1$), é que essa inclusão tenha apenas alterado a ordenação das arestas candidatas e que o nó n não tenha sido selecionado para rotear a mensagem. Nesse caso, pelo lema 4.1, prova-se que o teorema é verdadeiro.

Outro caso possível ($CASO2.2$), é que o nó n é selecionado para rotear a mensagem. Nesse ponto, caso o nó n roteie a mensagem para um nó pertencente a um caminho

válido, pelo algoritmo, o nó n será marcado como visitado e o roteamento continua para $(n-1)$ nós. Caso em que, pelo lema 4.1, o teorema é verdadeiro, pois o nó n é marcado como visitado e não é mais selecionado para roteamento. Caso o nó roteie a mensagem apenas para nós que não façam parte de um caminho válido, que não inclua nós visitados anteriormente pela mensagem, a mensagem, esta retornará ao nó n até que este fique sem alternativas de roteamento. Nesse ponto, o nó n retorna a mensagem ao nó que a enviou e, como é marcado como nó visitado, não será mais escolhido para o roteamento da mensagem. Portanto, o roteamento continua com, no máximo, $(n-1)$ nós, caso em que o teorema é válido, pois mesmo que tenha sido mudada a ordenação das arestas candidatas, o lema 4.1 prova que o algoritmo continua sendo válido.

Logo, conclui-se que a mensagem chega ao destino t, provando que o teorema é válido para qualquer número de nós. □

Capítulo 5

Resultados Simulados

Neste capítulo, o algoritmo robusto de roteamento proposto é avaliado com base em resultados obtidos através de simulações. Estas simulações foram efetuadas visando comparar as principais características desse algoritmo relativamente ao algoritmo de roteamento baseado no algoritmo de Dijkstra [15] e também ao algoritmo Schroeder-Duarte, apresentado no capítulo 3.

Para efetuar estas simulações, foi construído um simulador que permite a comparação de características específicas dos algoritmos sob um mesmo contexto de execução. Isto é, o simulador provê um protocolo básico para a geração e manutenção das topologias nos nós da rede, o qual é utilizado pelos três algoritmos comparados. Além desse protocolo básico, o simulador utiliza arquivos de entrada para informar a topologia da rede, as mensagens a serem enviadas e as situações de falhas. Com isso, é possível avaliar a execução desses algoritmos sob as mesmas condições de funcionamento.

O simulador implementado produz arquivos de *logs* dos resultados obtidos. Esses arquivos armazenam os detalhes de todo o processamento efetuado nas simulações. Detalhes como a troca de mensagens para atualização das topologias, os resultados de avaliações de arestas, a geração de tabelas de roteamento e os caminhos percorridos pelas mensagens.

Para validar a representatividade dos resultados obtidos, foi utilizada uma metodologia estatística baseada em intervalos de confiança. Essa metodologia mensura a probabilidade dos valores obtidos representarem a realidade, para os tipos de redes considerados. As

simulações foram efetuadas de modo a garantir que essa probabilidade esteja dentro de limites aceitáveis (a meia amplitude dos intervalos de confiança de 95% é menor que 10% da média dos valores obtidos).

Essa metodologia é explicada em detalhes na seção 5.2. O funcionamento do simulador é detalhado na seção 5.1. Por fim, a seção 5.3 apresenta os resultados obtidos juntamente com suas avaliações.

5.1 Simulador Desenvolvido

Com o intuito de comparar o algoritmo de roteamento proposto ao algoritmo de Dijkstra [15] e ao algoritmo descrito no capítulo 3, foi desenvolvido um simulador que executa esses algoritmos sob um mesmo contexto de execução. Esse contexto refere-se ao modo como é feita a atualização das topologias em cada nó da rede, ao grafo utilizado, às mensagens encaminhadas, ao conjunto de falhas utilizado e ao conjunto de parâmetros de execução do simulador.

A manutenção da topologia em cada nó da rede é efetuada de forma semelhante na simulação dos três algoritmos os três algoritmos. São utilizados basicamente os procedimentos `TopologyMaintainer`, `ReceiveUpdateMessage` e `ReceiveUpdateAckMessage`, descritos no capítulo 4, para desempenhar essa tarefa.

Vale ressaltar que o algoritmo Schroeder-Duarte não utiliza tabelas de roteamento, portanto o procedimento `TopologyMaintainer` não gera tabela alguma para esse algoritmo. Esse algoritmo também executa um procedimento distinto na manutenção de topologia: quando uma mensagem necessita ser retrocedida a um nó anterior já percorrido (isso ocorre quando as topologias dos nós estão desatualizadas devido à descoberta de uma falha), o nó que está enviando a mensagem a devolve ao nó anterior, antes enviando uma mensagem de atualização de topologia e, em seguida, a mensagem original. Dessa forma, existem mensagens de atualização de topologia que são propagadas antecipadamente nesse algoritmo.

Outro ponto importante para a comparação dos algoritmos é a criação e utilização dos arquivos que contém a topologia, o conjunto de mensagens e o conjunto de falhas. Esses

NODE N10	EDGE N12 N7
NODE N11	EDGE N1 N2
NODE N12	EDGE N1 N3
NODE N7	EDGE N3 N5
NODE N6	EDGE N5 N8
NODE N9	EDGE N8 N7
NODE N8	EDGE N2 N3
NODE N1	EDGE N2 N4
NODE N0	EDGE N4 N1
NODE N3	EDGE N4 N5
NODE N2	EDGE N4 N6
NODE N5	EDGE N6 N3
NODE N4	EDGE N6 N5
EDGE N0 N1	EDGE N6 N9
EDGE N0 N2	EDGE N9 N8
EDGE N0 N10	EDGE N9 N7
EDGE N10 N11	EDGE N6 N8
EDGE N11 N12	EDGE N9 N5

Figura 5.1: Exemplo de um arquivo de grafo.

arquivos são utilizados nas simulações, porém, caso não existam, são gerados em tempo de execução.

Ao iniciar uma simulação sem fornecer um arquivo com o grafo da rede, esse arquivo é gerado aleatoriamente, segundo a distribuição *power law* [20], para um número de nós definido por parâmetro ao simulador. Esse arquivo contém as indicações de nós e arestas do grafo, como mostra a figura de exemplo 5.1, e é a partir desse arquivo que são gerados os arquivos de mensagens e de falhas.

O arquivo de mensagens indica as mensagens de *payload* enviadas durante a simulação. Cada linha desse arquivo contém as informações para o envio de uma mensagem: o tempo de envio, o nó que gera a mensagem e o nó de destino. Um trecho de exemplo desse arquivo é mostrado na figura 5.2.

Durante a simulação, as mensagens descritas pelo arquivo de mensagens são enviadas. Porém, caso não seja fornecido um arquivo de mensagens ou caso sejam enviadas todas as mensagens descritas no arquivo antes que a condição de parada do simulador seja atingida, são geradas mensagens aleatórias até que a simulação termine. Essas mensagens geradas são gravadas no arquivo para que o próximo algoritmo a ser comparado utilize o mesmo conjunto de mensagens. Na geração de uma mensagem, o tempo de envio da mensagem

491700 N137 N34	491750 N34 N48
491800 N82 N40	491850 N16 N6
491900 N6 N86	491950 N141 N34
492000 N74 N77	492050 N11 N91
492100 N120 N105	492150 N90 N86
492200 N110 N89	492250 N84 N22
492300 N56 N34	492350 N80 N41
492400 N78 N148	492450 N65 N118
492500 N84 N99	492550 N68 N17
492600 N41 N84	492650 N75 N99

Figura 5.2: Trecho de exemplo de um arquivo de mensagens.

depende do intervalo de tempo entre mensagens (informado por parâmetro ao simulador), e os nós de envio e de destino são gerados aleatoriamente.

Para uma comparação adequada dos algoritmos, também é utilizado o arquivo de falhas. Esse arquivo indica, inicialmente, o tempo em que cada nó inicia a sua execução e, em seguida, as falhas que ocorrerão durante a simulação. Cada linha desse arquivo contém as informações de uma falha: o nó em que ocorre a falha (os resultados obtidos contemplam somente falhas de nós), o tempo inicial da falha e o tempo final da falha. Quando o tempo inicial é zero, o tempo final corresponde ao tempo em que o nó inicia a sua execução. Um trecho de exemplo desse arquivo é mostrado na figura 5.3.

Durante a simulação, os nós indicados neste arquivo são simulados como falhos. Porém, como no arquivo de mensagens, caso não seja fornecido um arquivo de falhas ou caso ocorram as falhas descritas no arquivo antes que a condição de parada do simulador seja atingida, são geradas falhas aleatórias até que a simulação termine. De forma similar, essas falhas geradas são gravadas no arquivo para que o próximo algoritmo a ser comparado utilize o mesmo conjunto de falhas. Na geração de uma falha, o nó que falha é definido aleatoriamente, e os tempos de início e de fim da falha dependem do intervalo de tempo de falhas (informado por parâmetro ao simulador).

Esse conjunto de arquivos/funcionalidades permite que os algoritmos sejam comparados dentro de um mesmo contexto de execução. E, para garantir que essa comparação é fidedigna, são produzidos *logs* de execução dos algoritmos. Esses logs armazenam todos os detalhes de uma execução: a troca de mensagens para atualização das topologias, os resultados de avaliações de arestas, a geração de tabelas de roteamento e os caminhos per-

N0 0 1571	N10 180000 180999
N1 0 13256	N11 181000 181999
N10 0 5947	N12 182000 182999
N11 0 18702	N10 183000 183999
N12 0 7367	N11 184000 184999
N2 0 12729	N12 185000 185999
N3 0 10847	N10 186000 186999
N4 0 2418	N11 187000 187999
N5 0 14737	N10 188000 188999
N6 0 12797	N11 189000 189999
N7 0 872	N12 190000 190999
N8 0 16700	N11 191000 191999
N9 0 13374	...

Figura 5.3: Trecho de exemplo de um arquivo de falhas.

Nó de destino	**Seqüência ordenada dos nós adjacentes**
N1	N1, N2, N10
N2	N2, N1, N10
N3	N1, N2, N10
N4	N1, N2, N10
N5	N1, N2, N10
N6	N1, N2, N10
N7	N1, N2, N10
N8	N1, N2, N10
N9	N1, N2, N10
N10	N10, N1, N2
N11	N10, N1, N2
N12	N10, N1, N2

Figura 5.4: Trecho de um *log* gerado pelo nó $N0$ da rede representada na figura 5.1.

corridos pelas mensagens. Com isso, pode-se não apenas observar mas também validar os resultados obtidos.

Esses *logs* ficam armazenados em dois tipos de arquivos: os *logs* da execução de cada nó, e o *log* dos caminhos percorridos pelas mensagens *payload*. A figura 5.4 mostra a tabela de roteamento gravada no *log* de um nó após a execução do procedimento `TopologyMaintainer` na simulação do algoritmo proposto neste trabalho. A figura 5.5 apresenta um trecho do *log* de mensagens.

Como pode-se observar na figura 5.5, o tempo de execução do nó em que a mensagem foi recebida é representado por um intervalo. Esse intervalo tem início quando é executado o procedimento manutenção de topologia (e geração de tabela, nos algoritmos que utilizam tabelas de roteamento), e termina antes da próxima execução desse procedimento. Ou

---

::PayloadMessageReceived- NodeId: N105 NodeTime: 283643-293642
MessageTime: 293100
Source: N1
Destination: N105
FullPath: $N1- > N0- > N44- > N66- > N105$
MainPath: $N1- > N0- > N44- > N66- > N105$
VisitedNodes: $N1- > N0- > N44- > N66- > N105$

::PayloadMessageReceivedWithBacktracking- NodeId: N142 NodeTime: 292014-293013
MessageTime: 293200
Source: N139
Destination: N142
FullPath: $N139- > N34- > N96- > N34- > N3- > N2- > N1- > N0- > N6- > N10- > N32- > N142$
MainPath: $N139- > N34- > N3- > N2- > N1- > N0- > N6- > N10- > N32- > N142$
VisitedNodes: $N139- > N34- > N96- > N3- > N2- > N1- > N0- > N6- > N10- > N32- > N142$

Figura 5.5: Trecho de um *log* de mensagens.

seja, uma mensagem só é encaminhada quando o seu tempo estiver dentro do respectivo intervalo de tempo dos nós que processam essa mensagem.

Além destes arquivos utilizados pelo simulador, este precisa de uma série de parâmetros para indicar o seu funcionamento. Esses parâmetros são descritos na tabela 5.1.

Parâmetro	**Descrição**
_NODES_NUMBER	Número de nós da rede
_ALPHA	Intervalo de tempo entre execuções do procedimento de manutenção de topologia
_BETA	Intervalo de tempo até considerar que um nó está falho
_MAX_START_NODES_TIME	Tempo máximo de inicialização dos nós na rede
_INIT_FAILS_TIME	*Tempo em que começam a ocorrer falhas na rede
_FAIL_INTERVAL	*Intervalo entre a ocorrência de falhas
_INIT_SEND_MESSAGES_TIME	*Tempo em que as mensagens começam a serem enviadas
_TIME_BETWEEN_MESSAGES	*Intervalo entre o envio de mensagens
_STOP_NUMBER	Condição de parada da simulação (em número de mensagens recebidas)

* Indica os parâmetros influenciam apenas a geração de arquivos de entrada.

Tabela 5.1: Descrição dos parâmetros do simulador.

As simulações efetuadas para avaliação dos algoritmos consideraram os seguintes valores em milissegundos: _ALPHA: 10000, _BETA: 3*_ALPHA, _MAX_START_NODES_TIME: 20000, _INIT_FAILS_TIME: 180000, _FAIL_INTERVAL: 10000, _INIT_SEND MESSAGES TIME: 180000 e _TIME_BETWEEN_MESSAGES: 50.

5.2 Metodologia de Simulação

Para validar a representatividade dos resultados produzidos, foi utilizada a técnica estatística de replicação [48]. Essa técnica baseia-se na repetição de simulações (replicação), usando sequências distintas de números randômicos para produzir o resultado de cada simulação. Neste caso, são utilizados grafos, mensagens e falhas geradas aleatoriamente em cada replicação.

Cada replicação produz m valores amostrados. As simulações efetuadas utilizaram m = 50.000, exceto nos casos em que foram analisados os efeitos da variação de m. Esses m valores são representados por uma média Y. O conjunto de k replicações produz $Y_1, Y_2, .., Y_k$, onde Y_k é a média dos m valores amostrados na replicação k. Cada Y_k é independente, pois foi produzido por uma rede gerada aleatoriamente para a replicação.

O objetivo das replicações é estimar o intervalo de confiança [13] da média $\overline{Y}$, sendo que $\overline{Y}$ é a média das médias Y_k das replicações efetuadas. Ou seja: $\overline{Y} = \sum_{i=1}^{k} \frac{Y_i}{k}$. E o intervalo de confiança é representado por: $\overline{Y} \pm H$, sendo H a semi-amplitude do intervalo.

Para se obter os resultados apresentados na próxima seção, foram efetuadas replicações das simulações até que se estimasse uma semi-amplitude menor que 10% da média $\overline{Y}$ para um intervalo de confiança de 95%. Como os resultados apresentados são as médias $\overline{Y}$ obtidas, isso significa que esses resultados representam os valores populacionais com uma confiança (probabilidade) de 95% para o seguinte intervalo: $\overline{Y} \pm 0.1 * \overline{Y}$.

5.3 Resultados Experimentais

Para a obtenção dos resultados apresentados a seguir, foram realizadas várias simulações para atender aos requisitos mencionados na seção anterior. Inicialmente, o algoritmo pro-

posto neste trabalho é comparado ao algoritmo de Dijkstra considerando os seguintes aspectos: comprimento médio de caminhos percorridos por mensagens entregues; e número de mensagens descartadas. Em seguida, na comparação com o algoritmo descrito no capítulo 3, são considerados os seguintes aspectos: comprimento médio dos caminhos percorridos por mensagens entregues; comprimento médio dos caminhos percorridos por mensagens entregues que efetuaram *backtracking*; número de mensagens descartadas; quantidade efetuada de cálculos de avaliação de aresta.

Todas comparações efetuadas nesse capítulo tiveram como objetivo principal a avaliação dos algoritmos em situações dinâmicas de funcionamento, i.e., situações em que ocorrem falhas, uma falha por vez, e essas falhas ocorrem por períodos relativamente curtos de 10 segundos. Esses períodos são curtos o suficiente para não serem percebidos nas atualizações normais de topologia, visto que no algoritmo anterior existem casos específicos em que a topologia nos nós é atualizada nesses casos.

Produzindo falhas por períodos curtos, consegue-se apurar principalmente as situações de interesse para avaliar a capacidade do algoritmo proposto de contornar as falhas encontradas no percurso das mensagens. Quando utilizam-se falhas por períodos prolongados, as topologias são alteradas e, quando o elemento falho retorna a funcionar, algumas mensagens podem ser descartadas desnecessariamente até que a topologia convirja novamente.

Outros pontos em comum nas comparações foram o intervalo entre envio de mensagens, o tempo de início de envio de mensagens e o tempo de início da ocorrência de falhas. O inicio do envio de mensagens e de ocorrência de falhas foi marcado para ocorrer no instante 180000, que representa 180 segundos após o início do funcionamento da rede. Esse tempo é superior à latência de convergência das redes simuladas. Já o intervalo de envio de mensagens foi determinado como sendo 50 milissegundos, o que gera uma quantidade de 200 mensagens aleatórias na rede durante a ocorrência de cada falha.

Dadas essas considerações iniciais, pode-se avaliar a figura 5.6. Essa figura apresenta o comprimento médio dos caminhos percorridos por mensagens entregues em redes de 150, 200 e 250 nós, do algoritmo proposto nesse trabalho e do algoritmo de Dijkstra. Pode-se observar que não houve uma alteração significativa nessa característica, o algoritmo

proposto percorreu, em média, aproximadamente um nó a mais por mensagem entregue.

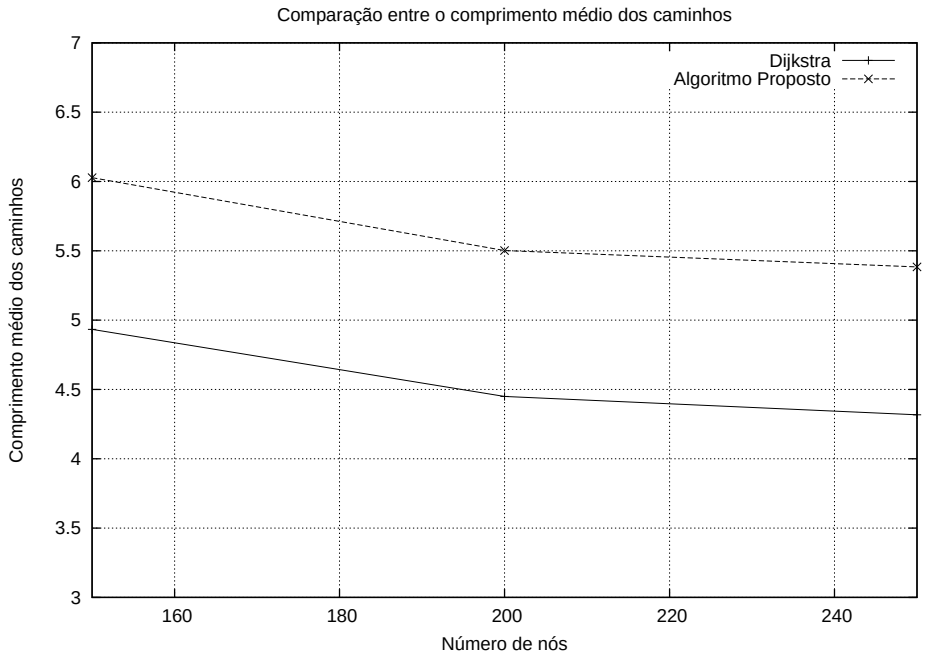

Figura 5.6: A diferença no comprimento médio dos caminhos entre o algoritmo proposto e o algoritmo de Dijkstra.

Observa-se, ainda, que o resultado do algoritmo proposto inclui mensagens entregues que contornaram falhas ocorridas na rede e que percorreram um caminho maior por esse motivo. Essas mensagens foram descartadas pelo algoritmo de roteamento baseado no algoritmo de Dijkstra. O número exato de mensagens descartadas pode ser observado na figura 5.7. Esta figura compara o número de mensagens descartadas na execução do algoritmo proposto e do algoritmo de roteamento baseado no algoritmo de Dijkstra.

Nota-se que o algoritmo proposto reduziu significativamente o número de mensagens descartadas. De fato, nos casos avaliados, o algoritmo proposto descartou apenas mensagens que não tinham possibilidade de roteamento, devido ao particionamento das redes pelas falhas inseridas na simulação. Entretanto, existem situações, como no caso de topologias desatualizadas descrito no início desta seção, em que o algoritmo proposto pode descartar uma mensagem mesmo que haja um caminho possível. Nesses casos, tais mensagens seriam descartadas pelos três algoritmos simulados.

Quando comparado ao algoritmo Schroeder-Duarte [50], o algoritmo proposto quase não apresenta diferença quando avalia-se o comprimento médio dos caminhos percorridos

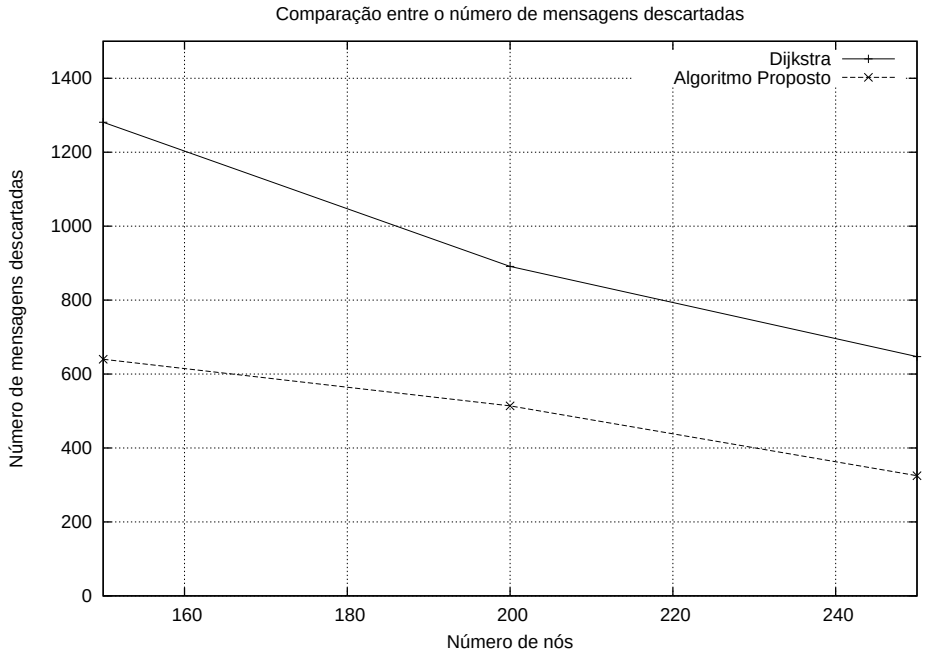

Figura 5.7: A diferença no número de mensagens descartadas entre o algoritmo proposto e o algoritmo de Dijkstra.

por mensagens. A figura 5.8 apresenta essa diferença para redes de 150, 200 e 250 nós. Porém, quando analisados os resultados apresentados pela figura 5.9, pode-se constatar uma diferença maior no comprimento médio de caminhos de mensagens que efetuaram o *backtracking*.

Essa diferença deve-se ao fato do algoritmo Schroeder-Duarte enviar uma mensagem de atualização de topologia ao nó imediatamente anterior, nos casos em que é necessário efetuar o *backtracking* de uma mensagem. Isto faz com que o nó anterior atualize a sua representação da topologia da rede e, portanto, roteie a mensagem por um caminho mais adequado.

Esta característica de adiantar as mensagens de atualização de topologia, em casos de descoberta de falhas na rede, pode ser prejudicial quando o elemento falho retorna ao seu estado normal. Este fato pode ser observado na figura 5.10, que compara a quantidade de mensagens descartadas pelos algoritmo Schroeder-Duarte e pelo algoritmo proposto. Ao retomar o seu estado de funcionamento normal, as mensagens somente serão roteadas a esses elementos quando as topologias dos demais nós pertencentes ao caminho normal da mensagem forem atualizadas. No pior caso, esse tempo de convergência está vinculado ao

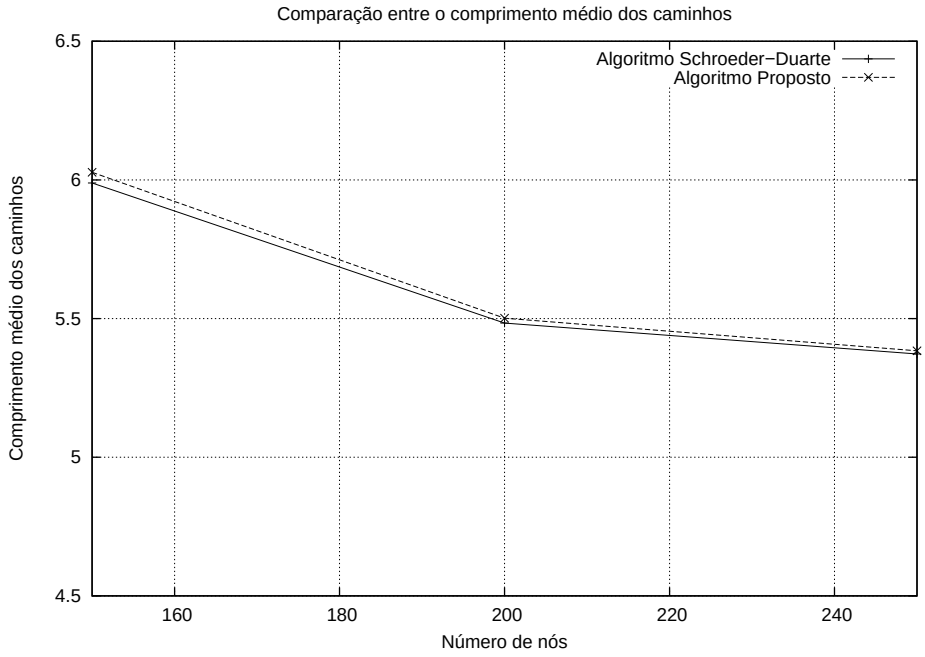

Figura 5.8: A diferença no comprimento médio dos caminhos entre o algoritmo proposto e o algoritmo Schroeder-Duarte.

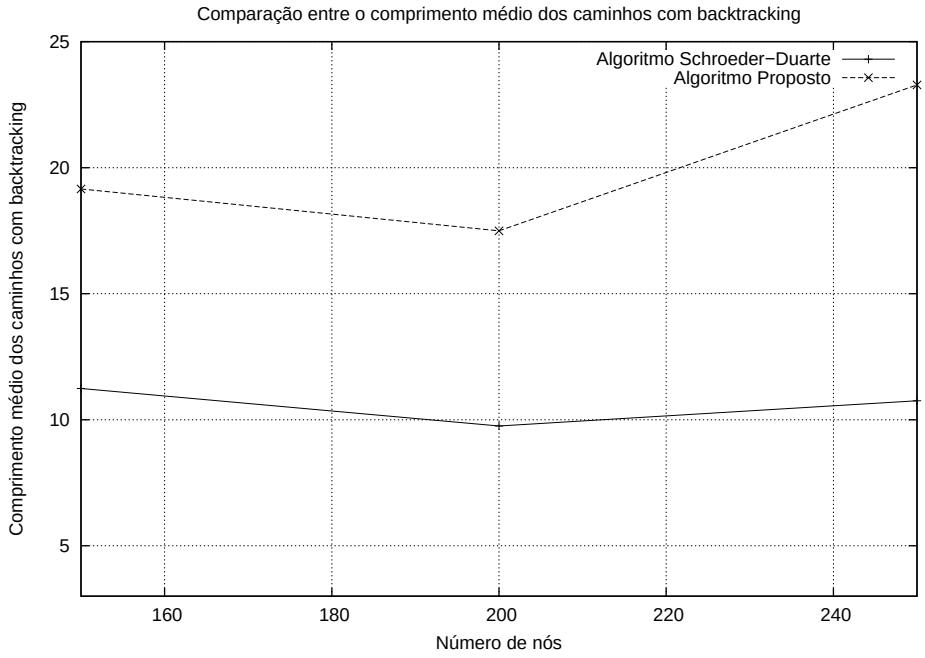

Figura 5.9: A diferença no comprimento médio dos caminhos com *backtracking* entre o algoritmo proposto e o algoritmo Schroeder-Duarte.

diâmetro da rede em questão.

Outro ponto importante na comparação com o algoritmo Schroeder-Duarte é em relação ao processamento efetuado durante o roteamento. Para isso, foi medida a quantidade efetuada de cálculos de avaliação de arestas. Essa quantidade pode ser observada

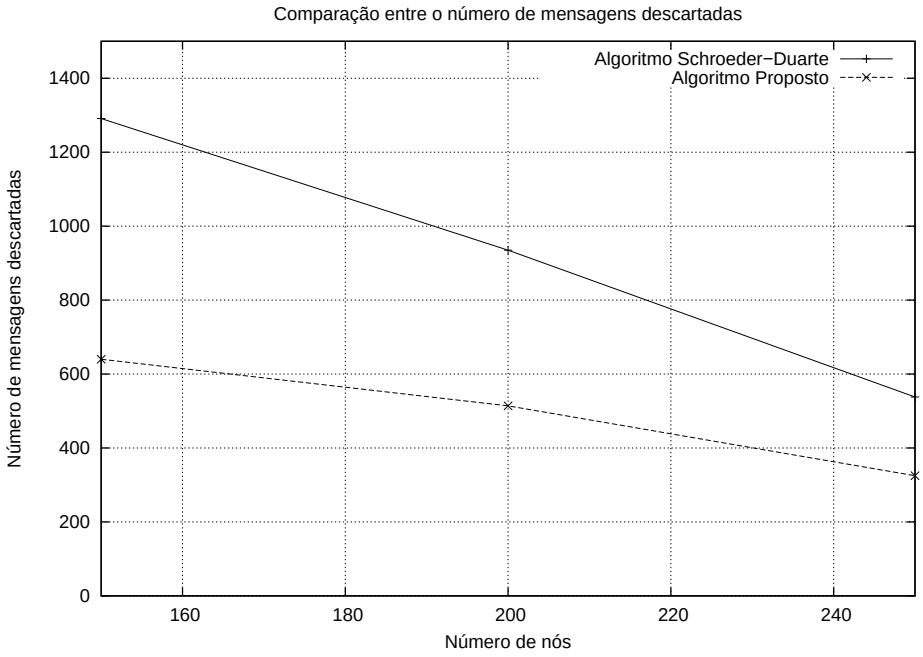

Figura 5.10: A diferença no número de mensagens descartadas entre o algoritmo proposto e o algoritmo Schroeder-Duarte.

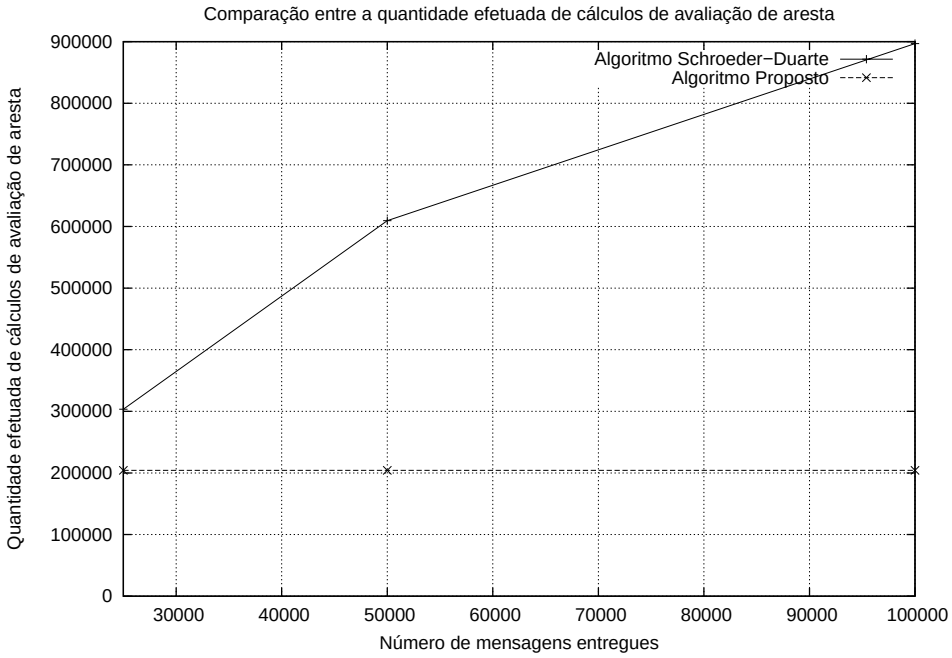

Figura 5.11: A diferença na quantidade efetuada de cálculos de avaliação de aresta entre o algoritmo proposto e o algoritmo Schroeder-Duarte.

na figura 5.11. Enquanto os resultados apresentados nessa figura mostram claramente a dependência do número de mensagens encaminhadas no desempenho do algoritmo apresentado no capítulo 3, os resultados apresentados pelo algoritmo proposto mostram-se dependentes unicamente do processo de geração de tabelas de roteamento.

Para um número pequeno de mensagens o algoritmo Schroeder-Duarte efetua um processamento inferior ao processamento imposto pelo algoritmo proposto. Porém, o aumento do número de mensagens encaminhadas faz com que o processamento necessário ao encaminhamento de mensagens no algoritmo algoritmo inicial seja superior ao processamento necessário à manutenção das tabelas de roteamento no algoritmo proposto.

Capítulo 6

Conclusão

Neste trabalho foi proposto um algoritmo heurístico de roteamento cujo objetivo é o de possibilitar a elaboração prática de um protocolo de roteamento tolerante a falhas. Esse algoritmo permite criação de tabelas de roteamento, baseadas na avaliação conjunta das arestas segundo os critérios de fluxo máximo e caminho mínimo, que possibilitam o *backtracking* (*loop* intencional) de mensagens durante o processo de encaminhamento. Juntamente com a descrição desse algoritmo, foi apresentada uma prova formal de sua correção. Foram também apresentados resultados de simulações desse algoritmo, comparando-o aos algoritmos de Dijkstra [15] e Schroeder-Duarte [50]. Esses resultados comparam características como o comprimento médio de caminhos percorridos, comprimento médio de caminhos que efetuaram *backtracking*, a quantidade efetuada de cálculos avaliação de aresta, e o número de mensagens descartadas para redes, com topologia *power law*, de 150 à 250 nós.

Trabalhos futuros contemplam a utilização do critério de fluxo máximo junto a outros critérios, como os de QoS, e sua avaliação para a geração das tabelas de roteamento. A elaboração de um protocolo que utilize esse algoritmo em redes reais também faz-se necessária.

Referências Bibliográficas

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. *Resilient overlay networks*. ACM Press, 2001.

[2] A. Basu and J. Riecke. Stability issues in OSPF routing. *Proceedings of the 2001 SIGCOMM conference*, 31(4):225–236, 2001.

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

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475: An Architecture for Differentiated Service. *Internet RFCs*, 1998.

[5] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overwiew*, 1994.

[6] A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved BGP convergence via ghost flushing. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2.

[7] S. Bryant, M. Shand, and S. Previdi. IP Fast Reroute Using Not-via Addresses. *IETF Internet Draft*, 2006.

[8] E. Calle, J. L. Marzo, A. Urra, and P. Vila. Enhancing MPLS QoS routing algorithms by using the network protection degree paradigm. *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, 6, 2003.

[9] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A Framework for Multi-Protocol Label Switching. *Work in Progress*, 1999.

[10] Douglas E. Comer. *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architectures*. Prentice Hall PTR, 2006.

[11] T.H. Cormen, C. Leiserson, R.L. Rivest, and C. Stein. *Algoritmos: Teoria e Prática*. 2002.

[12] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. *RFC2386: A Framework for QoS-based Routing in the Internet*, 1998.

[13] J.L. Devore et al. *Probability and statistics for engineering and the sciences*. Brooks/Cole Publishing Company, 1995.

[14] R. Diestel. *Graph Theory*. Springer, 2005.

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

[16] K. Downes et al. *Internetworking Technologies Handbook*. Cisco Press Indianapolis, Ind, 1998.

[17] 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, 2004.

[18] P. Echenique, J. Gomez-Gardenes, and Y. Moreno. Improved routing strategies for Internet traffic delivery. *Physical Review E*, 70(5):56105, 2004.

[19] Cicic et. al. Relaxed Multiple Routing Configurations for IP Fast Reroute. *NOMS 2008*, pages 457–464, 2008.

[20] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262, 1999.

[21] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.

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

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

[24] M. Gjoka, V. Ram, and X. Yang. Evaluation of IP Fast Reroute Proposals. *IEEE International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE)*.

[25] J. Gross and J. Yellen. *Graph theory and its applications*. CRC Press, Inc., 1999.

[26] S. Halabi et al. *Internet Routing Architectures*. Cisco Press Indianapolis, IN, 2000.

[27] C. L. Hedricks. *RFC 1058: Routing Information Protocol*, 1988.

[28] J. Hershberger and S. Suri. Vickrey prices and shortest paths: what is an edge worth? *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 252–259, 2001.

[29] C. Huang, V. Sharma, K. Owens, and S. Makam. Building reliable MPLS networks using a path protection mechanism. *Communications Magazine, IEEE*, 40(3):156–162, 2002.

[30] C. Huitema. *Routing in the Internet*. Prentice-Hall, Inc., 1995.

[31] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP Network Recovery Using Multiple Routing Configurations. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–11, 2006.

[32] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *Networking, IEEE/ACM Transactions on*, 9(3):293–306, 2001.

[33] W. Li. Inter-domain routing: Problems and solutions. *State University of New York at Stony Brook*, 2003.

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

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

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

[37] G. Malkin. *RFC 1388: RIP version 2 carrying additional information, January 1993.*

[38] G. Malkin. *RFC 1387: RIP Version 2 Protocol Analysis*, 1993.

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

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

[41] D. Mills. *RFC 1772: Exterior Gateway Protocol Formal Specification*, 1984.

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

[43] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the 31st ACM symposium on Theory of computing*, pages 129–140. ACM Press, 1999.

[44] D. Oran. *OSI IS-IS Intra-domain Routing Protocol (RFC 1142)*, 1990.

[45] D. Pei, M. Azuma, D. Massey, and L. Zhang. BGP-RCN: Improving BGP convergence through root cause notification. *Computer Networks*, 48(2):175–194, 2005.

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

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

[48] SM Sanchez. ABC's of output analysis. *Simulation Conference Proceedings, 1999. Winter*, 1, 1999.

[49] 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.

[50] J. Schroeder and E. P. Duarte Jr. Roteamento Dinâmico Tolerante a Falhas Baseado em Avaliação de Fluxo Máximo. *Workshop de Testes e Tolerância a Falhas (WTF'2006), Anais do SBRC'2006*, pages 197–207, 2006.

[51] G. Yan, T. Zhou, B. Hu, Z.Q. Fu, and B.H. Wang. Efficient routing on complex networks. *Physical Review E*, 73(4):46108, 2006.