# Um Algoritmo para *Fast-ReRoute* Utilizando Avaliação de Fluxo Máximo e *Backtracking*

**Leon Okida, Elias P. Duarte Jr.**

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

{laog19,elias}@inf.ufpr.br

***Resumo.*** *Este trabalho apresenta o algoritmo MaxFlowRouting para o roteamento tolerante a falhas. O algoritmo propõe uma estratégia de Fast-ReRoute (FRR) para lidar com falhas na rede. Um roteador mantém uma rota primária e rotas alternativas para cada destino. Quando uma falha é detectada na rota primária, uma rota alternativa é empregada para contornar o ponto em que ocorre a falha. O algoritmo MaxFlowRouting utiliza a avaliação de fluxo máximo para computar rotas que possuem mais opções de rotas alternativas que possam ser ativadas em caso de falha. Se não há rota alternativa funcional, o algoritmo faz backtracking para o roteador anterior. Resultados experimentais obtidos com simulação confirmam que o algoritmo MaxFlowRouting produz rotas com até 1,54 vezes mais rotas alternativas por vértice.*

## 1. Introdução

Organizações e indivíduos estão cada vez mais dependentes das redes de computadores, particularmente da Internet. Em algumas instâncias, a Internet pode apresentar bastante instabilidade [Bischof et al. 2023, Duarte Jr et al. 2010]. Para aumentar a robustez da rede nesses casos, deve-se adotar estratégias de roteamento com tolerância a falhas [Liu et al. 2024, Duarte Jr et al. 2004]. Isso contribui para que a rede se recupere rapidamente da ocorrência de uma falha em sua infraestrutura [Duarte and Musicante 1999]. O *Fast-ReRoute* (FRR) [Shand and Bryant 2010] é uma estratégia proativa de reparo em caso de falhas, em que uma rota alternativa é empregada quando a rota primária falha. O FRR foi adotado em diversos protocolos da Internet, como o IP [Shand and Bryant 2010], o MPLS [Pan et al. 2005] e o OSPF [Filsfils et al. 2012].

A estratégia alternativa ao FRR para reparação de falhas é reativa e consiste em esperar as tabelas de roteamento reconvergirem para a nova topologia da rede. Isso causa um período de disrupção no tráfego da rede, em que pacotes já enviados ao destino através da rede anteriormente à falha serão descartados, causando, consequentemente, prejuízos para serviços distribuídos e aplicações. O FRR computa rotas alternativas que são armazenadas na tabela de roteamento junto da rota principal. Assim, quando um roteador detecta uma falha, a rota alternativa pode ser empregada imediatamente. O período de disrupção é reduzido para o tempo que o roteador leva para detectar a falha e ativar a rota alternativa. Entretanto, a efetividade do FRR depende da existência de rotas alternativas que possam contornar o ponto em que ocorreu a falha.

Neste trabalho, é proposto o algoritmo *MaxFlowRouting*, baseado em avaliação do fluxo máximo [Cormen et al. 2022, Schroeder et al. 2004], para a seleção de rotas robustas. A avaliação de fluxo máximo encontra, implicitamente, rotas com maior conectividade possível considerando a topologia da rede. Isso acontece porque o fluxo máximo

é igual à quantidade de caminhos disjuntos em arestas. Consequentemente, é maior o número de rotas alternativas que podem ser utilizadas na ocorrência de uma falha.

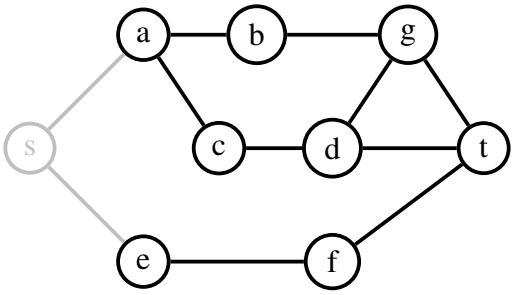


**Figura 1. Exemplo de escolha do próximo *hop* baseada em avaliação do fluxo máximo.**

O algoritmo *MaxFlowRouting* é executado por um roteador e constrói uma tabela de roteamento com múltiplas alternativas para o próximo *hop* para cada destino. A Figura 1 ilustra como um roteador executando o algoritmo *MaxFlowRouting* escolhe o próximo *hop*. Na figura, a origem $s$ precisa enviar uma mensagem ao destino $t$. O algoritmo está sendo executado pelo vértice $s$. Existem duas arestas adjacentes a ele: $(s, a)$ e $(s, e)$. Isso significa que o próximo *hop* será para $a$ ou para $e$. Para fazer essa escolha, o algoritmo inicialmente gera um subgrafo removendo o vértice que está executando o algoritmo. Consequentemente, as arestas adjacentes a esse vértice são removidas também. Na figura, o subgrafo é formado pelas linhas e vértices escuros.

Para avaliar a opção pela aresta $(s, a)$, o algoritmo computa os critérios de caminho mínimo e de fluxo máximo entre $a$ e $t$ no subgrafo. Ele obtém, nesse caso, o caminho mínimo de tamanho 3 e o fluxo máximo de valor 2. Fazendo a mesma avaliação para a opção pela aresta $(s, e)$, obtém-se o caminho mínimo de tamanho 2 e o fluxo máximo de valor 1. Isso significa que existem dois caminhos disjuntos entre $a$ e $t$, enquanto apenas um entre $e$ e $t$. O algoritmo utiliza pesos para definir a influência de cada critério na decisão do próximo *hop*. Como o foco do algoritmo é tolerância a falhas, o critério de fluxo máximo deve ter um peso maior. No exemplo, o vértice $a$ tem preferência maior como o próximo *hop* por ter um valor de fluxo máximo do que o vértice $e$.

No trabalho, também é descrita uma estratégia de FRR baseada em *backtracking* que pode ser utilizada em qualquer tabela de roteamento que possua múltiplas alternativas de próximo *hop* por destino. Se a rota alternativa também falha, o pacote pode retornar para o roteador anterior para que ele tente outra rota. Assim, o roteamento tem maiores chances de funcionar, mesmo que as tabelas de roteamento não reflitam a topologia real da rede. Basta que nenhum roteador visitado anteriormente falhe e que exista pelo menos uma rota funcional entre a origem e o destino. Essa estratégia de FRR pode utilizar rotas produzidas tanto pelo algoritmo *MaxFlowRouting* como por outros algortimos de roteamento, como o algoritmo de Dijkstra [Cormen et al. 2022]. Destaca-se que em [Schroeder and Duarte Jr 2007] é descrito um algoritmo de roteamento que também utiliza fluxo máximo, entretanto aquele *não* é um algoritmo de re-roteamento rápido: cada mensagem carrega toda a rota já percorrida, e quando é recebida por um roteador, este atualiza a topologia removendo todos os vértices já percorridos (e todas as suas arestas adjacentes) e recalcula o próximo passo executando o algoritmo de fluxo máximo para

cada mensagem processada. O algoritmo apresentado no presente artigo reduz o custo ao executar o fluxo máximo apenas para pré-computar rotas alternativas.

O algoritmo proposto foi avaliado através de simulação e comparado com uma versão do FRR que calcula rotas com o algoritmo de Dijkstra [Okida et al. 2024]. Os dois algoritmos foram executados em grafos de Mundo Pequeno de Watts-Strogatz [Watts and Strogatz 1998] e em grafos correspondendo a *backbones* importantes da Internet, como a RNP do Brasil [RNP 2025], a Internet2 dos Estados Unidos da América [Internet2 2025], a Géant da Europa [GÉANT 2025] e a Wide do Japão [WIDE Project 2025]. Resultados confirmam que o algoritmo proposto produz rotas com mais opções de alternativas: no melhor caso, o *MaxFlowRouting* computou rotas com até 1,54 vezes mais rotas alternativas por vértice.

O restante deste trabalho está organizado como segue. A Seção 2 apresenta o algoritmo *MaxFlowRouting* e o algoritmo de FRR propostos neste trabalho. A Seção 3 apresenta os resultados de avaliação empírica. Por fim, a Seção 4 conclui o trabalho.

## 2. O Algoritmo *MaxFlowRouting*

Esta seção apresenta o algoritmo *MaxFlowRouting* para roteamento tolerante a falhas. Um roteador que executa o algoritmo mantém o grafo $G = (V, E)$ como uma representação local da topologia da rede. Não é necessário que o grafo $G$ reflita exatamente a topologia real da rede, o que é essencial tendo em vista os desafios para que reflete exatamente o estado completo da topologia real da rede [Nassu et al. 2007]. Quando um roteador detecta uma mudança na topologia, ele deve disseminar essa informação para os demais roteadores. A tabela de roteamento e o grafo $G$ são atualizados pelo roteador quando este recebe uma informação de mudança na topologia. Cada roteador executa o Algoritmo 1 (*MaxFlowRouting*) para gerar a tabela de roteamento para todos os destinos possíveis na rede. O algoritmo utiliza a avaliação de fluxo máximo e o algoritmo de Dijkstra para decidir qual deve ser o próximo *hop* da rota.

O fluxo máximo [Schroeder et al. 2004] é equivalente ao problema do corte mínimo [Cohen et al. 2011b, Cohen et al. 2012]. O corte mínimo $m$ entre dois vértices $u$ e $v$ de um grafo $G$ é o conjunto de arestas $C$, tal que a remoção de todas as arestas de $C$ em $G$ remove todos os caminhos possíveis entre $u$ e $v$, desconectando-os. A cardinalidade do corte $C$, denotada por $|C|$ é o número de arestas em $C$. Um corte $C$ é chamado de corte mínimo se, para cada corte $C'$ entre o mesmo par de vértices $u$ e $v$ no grafo $G$, $|C| \leq |C'|$. Para cada par de vértices no grafo $G$, o fluxo máximo e o corte mínimo possuem a mesma cardinalidade e podem ser computados pelos mesmos algoritmos [Cohen et al. 2017, Maske et al. 2020]. Neste trabalho, considera-se que a capacidade de fluxo de todas as arestas é 1. Assim, a avaliação de fluxo máximo encontra, implicitamente, rotas com mais conectividade e que possuem mais opções de caminhos alternativos em caso de falha. Isso ocorre porque o fluxo máximo entre dois vértices indica o número de caminhos disjuntos em arestas entre eles [Cohen et al. 2011a].

Um roteador seleciona o próximo *hop* ordenando todos os seus vizinhos com base em uma função avaliadora $\Gamma(G', i, d)$. A função é executada por cada roteador $s$ para cada vizinho $i$ considerando um destino $d$. Essa função utiliza o grafo $G' = (V', E')$ onde $V' = V - s$ e $E' = E - (s, j), \forall j | (s, j) \in E$. A função $\Gamma(G', i, d)$ retorna um valor numérico para cada vértice $i$ adjacente a $s$ considerando o destino $d$, de forma que valores

altos são considerados melhores que valores baixos, como descrito a seguir.

A função $\Gamma(G', i, d)$ é computada levando em conta o fluxo máximo e o tamanho da rota, sendo calculada pela expressão $\Gamma(G', i, d) = w_1 c_1(G', i, d) + w_2 c_2(G', i, d)$. Essa expressão leva em conta os critérios de valor do fluxo máximo entre $i$ e $d$ no grafo $G'$ (critério $c_1$) e de tamanho da rota de tamanho mínimo entre $i$ e $d$ no grafo $G'$ (critério $c_2$). Cada critério é multiplicado por um peso (respectivamente $w_1$ e $w_2$), que determina como o critério influencia o resultado da função $\Gamma(G', i, d)$. O peso correspondente ao critério de fluxo máximo deve ser positivo, pois o objetivo é maximizá-lo, enquanto o peso correspondente ao critério de caminho mínimo deve ser negativo, pois deseja-se minimizá-lo. Então, dado um vértice $s$, todos os seus vizinhos $i$ que possuem caminho para o destino $d$ são ordenados de modo decrescente de acordo com o resultado da função $\Gamma(G', i, d)$ como candidatos a próximo *hop*. Os vértices $i$ de maior valor de $\Gamma(G', i, d)$ são escolhidos primeiro para rotear um pacote de $s$ para o destino $d$.

O Algoritmo 1 (*MaxFlowRouting*) é executado por cada roteador $s$, que inicialmente se remove do grafo $G$ da topologia conhecida, gerando o grafo $G'$. Após isso, para cada destino possível $d$ e para cada vizinho $i$, $s$ computa o fluxo máximo e o tamanho do menor caminho entre $i$ e $d$. Feito isso, a função $\Gamma(G', i, d)$ é computada utilizando pesos para os critérios de fluxo máximo e distância. Para cada destino (entrada na tabela de roteamento), existe uma lista de *CandidatosAProximoHop*, que é inicializada vazia. Um vizinho $i$ é incluído em *CandidatosAProximoHop* se possui um caminho para o destino $d$ em $G'$. Finalmente, os vizinhos em *CandidatosAProximoHop* são ordenados de acordo com a função avaliadora $\Gamma(G', i, d)$.

**Algorithm 1** Especificação do algoritmo *MaxFlowRouting*: Geração da tabela de roteamento do roteador $s$

```
1: G' ← G − {s}
2: for all destino d em G' do
3:    CandidatosAProximoHop ← lista vazia
4:    for all vizinho i adjacente a s em G do
5:       if não há caminho entre i a d em G' then
6:          ignora i
7:       else
8:          adiciona i a CandidatosAProximoHop
9:          computa Γ(G', i, d)
10:      end if
11:   end for
12:   if CandidatosAProximoHop possui mais de um elemento then
13:      ordena CandidatosAProximoHop de acordo com Γ(G', i, d)
14:   end if
15:   TabelaDeRoteamento[d] ← CandidatosAProximoHop
16: end for
```

A tabela de roteamento gerada pelo algoritmo *MaxFlowRouting* é utilizada como base para o roteamento de pacotes. O Algoritmo 2 de FRR utiliza a tabela de roteamento para selecionar o próximo *hop* dado o destino do pacote.

O Algoritmo 2 (*Fast-ReRoute* com *Backtracking*) é executado pelo roteador $s$ para

encaminhar um pacote enviado pela origem $o$ para o destino $d$. Primeiro, o roteador $s$ checa se ele é vizinho do roteador $d$ e, caso seja, envia o pacote diretamente a $d$. Cada pacote $pct$ carrega consigo a lista sequencial dos vértices pelos quais passou, chamada de $pct.NodosVisitados$. O roteador $s$ adiciona a si mesmo a $pct.NodosVisitados$ caso não esteja na lista. Caso o roteador já esteja na lista, um ciclo é detectado, o que ocorre quando o pacote é enviado por *backtracking* por um vizinho que não tinha rota disponível para o destino. Após isso, o roteador $s$ seleciona o próximo *hop* para encaminhar o pacote. O vizinho de melhor classificação na entrada da tabela de roteamento para o destino $d$ que não esteja em $pct.NodosVisitados$ é selecionado para ser o próximo *hop*. Caso não exista um vizinho disponível para isso, o roteador $s$ retorna o pacote para o roteador $r$ de onde recebeu o pacote, isto é, faz *backtracking*. Caso não exista $r$ antes de $s$, então $s$ é própria a origem $o$ do pacote e não existe rota disponível entre $o$ e $d$.

**Algorithm 2** Especificação do algoritmo *Fast-ReRoute* com *Backtracking* no roteador $s$.

**if** destino $d$ é adjacente a $s$ em $G$ **then**
  **encaminha** pacote $pct$ para $d$
**else if** $pct.NodosVisitados$ não contém o roteador $s$ **then**
  **adiciona** $s$ a $pct.NodosVisitados$
  **if** $TabelaDeRoteamento[d]$ contém roteadores que não estão em $pct.NodosVisitados$ **then**
    **escolhe** o próximo *hop* $i$ de acordo com $TabelaDeRoteamento[d]$, tal que $i \notin pct.NodosVisitados$
    **envia** pacote $pct$ para $i$
  **else**
    {Não é possível encaminhar o pacote, deve fazer *backtracking*}
    **if** $pct.NodosVisitados$ contém pelo menos um roteador $r$ antes de $s$ **then**
      **envia** $pct$ de volta para $r$
    **else**
      {Não é possível fazer *backtracking*, pois está na origem $o$}
      **retorna** Erro: Não há rota disponível entre $o$ e $d$
    **end if**
  **end if**
**end if**

### 2.1. Estudos de Caso

As Figuras 2 e 3 apresentam dois exemplos de aplicação do algoritmo *MaxFlowRouting* e do algoritmo proposto de *Fast-ReRoute* com *Backtracking*. A parte (a) da figura apresenta a situação em que o enlace $(b, t)$ falha durante o roteamento de um pacote do nodo $s$ para o nodo $t$. Inicialmente, $s$ envia o pacote para $a$, que é a única opção possível para o próximo *hop*. Então, $a$ recebe o pacote e consulta sua tabela de roteamento para decidir para qual roteador deve encaminhá-lo. Na tabela de roteamento de $a$, o nodo $b$ é indicado como a opção preferencial de próximo *hop*, pois possui o maior valor atribuído pela função avaliadora $\Gamma$. Assim, $a$ encaminha o pacote para $b$. Então, o nodo $b$ tenta enviar o pacote diretamente para $t$, mas não consegue devido à falha no enlace $(b, t)$. Como não possui mais opções de próximo *hop*, $b$ retorna o pacote para $a$. Após receber o pacote novamente,

$a$ verifica então o nodo com o maior valor retornado pela função avaliadora $\Gamma$ que ainda não foi visitado. Assim, $a$ encaminha o pacote para o nodo $c$. A rota percorrida até este momento é $(s, a, c)$, com o nodo $b$ também marcado como visitado, mas sem fazer parte da rota utilizada para chegar ao destino.

A tabela de roteamento de $c$ apresenta as opções $a$, $h$ e $d$ ordenadas pela função avaliadora $\Gamma$. A opção prioritária é $a$, mas esta já está marcada como visitada. Portanto, o nodo $h$ é escolhido. O nodo $h$ possui um valor retornado por $\Gamma$ maior do que o nodo $d$ porque possui um tamanho menor de caminho mínimo até $t$, desconsiderando a falha no enlace $(b, t)$, que é desconhecida por $c$. O pacote é então encaminhado para $h$. Entretanto, as opções de roteamento a partir de $h$ já estão marcadas como visitadas. Por isso, o pacote é retornado para $c$, que escolhe então o nodo $d$. Em seguida, o pacote é encaminhado para $d$ e percorre o caminho $(d, e, f, g, t)$ para chegar até o destino final.

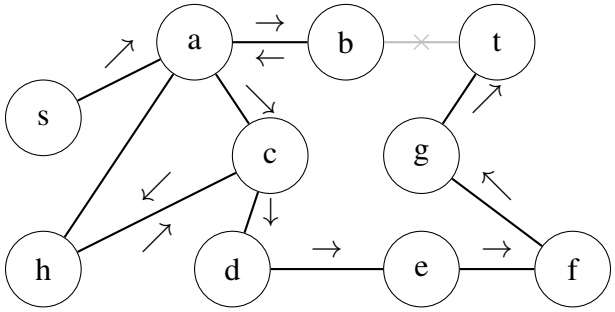


**Figura 2. Estudo de caso: O enlace $(b, t)$ falha quando $b$ tenta enviar o pacote para $t$. O algoritmo de FRR com *backtracking* é utilizado para rotear, com sucesso, o pacote para $t$.**

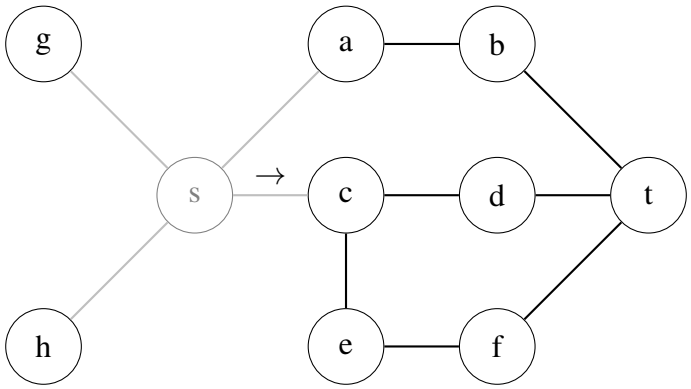


**Figura 3. Estudo de caso: O nodo $s$ computa os critérios de fluxo máximo e caminho mínimo para escolher o próximo *hop*. O nodo escolhido para o próximo *hop* é $c$.**

A Figura 3 mostra um segundo caso de estudo, que ilustra como um nodo seleciona um de seus vizinhos para o roteamento. O nodo $s$ roteia um pacote para o nodo $t$. Na topologia original, representada pelo grafo $G$, o nodo $s$ possui quatro opções para o próximo *hop*: os nodos $a, c, g$ e $h$. Para avaliar as opções com a função $\Gamma$, é gerado o grafo $G'$ retirando $s$ e todas as suas arestas adjacentes de $G$. Sobram, assim, apenas dois vértices em $G'$ que possuem pelo menos um caminho para $t$: $a$ e $c$. A função $\Gamma$ retorna um valor maior para o nodo $c$, pois este possui um valor maior para o critério de fluxo máximo e o mesmo valor para o critério de tamanho do caminho mínimo comparado ao

nodo $a$. Então, o nodo $s$ escolhe o nodo $c$ para encaminhar o pacote. O pacote então percorre a rota $(s, c, d, t)$ para chegar até o destino. Note que a rota que passa por $c$ possui mais opções de rotas alternativas para serem empregadas em caso de falha do que a única rota possível que passa por $a$.

### 2.2. Prova de Corretude do Algoritmo de FRR com *Backtracking*

Esta seção apresenta a prova de que o algoritmo de FRR com *backtracking* consegue rotear um pacote do nodo $s$ para o nodo $t$ pela rede representada pelo grafo $G$ se nenhum nodo visitado anteriormente falha e se existe pelo menos uma rota funcional da origem até o destino.

**Teorema 2.1.** Considere o grafo $G = (V, E)$ representando a topologia da rede e dois vértices selecionados arbitrariamente $s, t \in V$, representando a origem e o destino de um pacote $p$. O algoritmo de FRR com *backtracking* consegue rotear o pacote com sucesso entre $s$ e $t$ se nenhum vértice já visitado falha e se existe pelo menos uma rota funcional entre $s$ e $t$ em $G$.

**Prova.** O nodo executando o algoritmo de FRR com *backtracking* tenta enviar o pacote através do nodo vizinho de melhor avaliação na entrada da tabela de roteamento correspondente ao destino $t$. Este nodo, por sua vez, também faz isso. Esse processo se repete até que o pacote chegue ao destino. Caso um nodo $j$ receba o pacote e detecte que não possui uma rota funcional para o destino, ele pode retornar o pacote ao nodo $i$ anterior de onde recebeu o pacote. O nodo $i$ tenta enviar o pacote através do próximo nodo $k$ de melhor avaliação na entrada da tabela de roteamento correspondente ao destino. O processo se repete: se não há rota funcional entre $k$ e $t$, o pacote será novamente retornado para $i$. Após $i$ tentar todas as suas opções de próximo *hop* sem sucesso, ele retorna para o nodo anterior de onde recebeu o pacote. Se nenhum nodo já visitado anteriormente falhar, esse processo pode acontecer, sucessivamente, até o pacote ser retornado para a origem $s$, que também tentará todas as suas alternativas de próximo *hop*. Assim, se existe uma rota funcional entre a origem e o destino na rede, ela será encontrada após um tempo e utilizada para rotear o pacote com sucesso para o destino $t$. □

## 3. Avaliação Empírica

Esta apresenta os resultados experimentais obtidos com simulação comparando o algoritmo *MaxFlowRouting* com uma alternativa para FRR que usa o algoritmo de Djkstra para calcular as rotas. Os algoritmos foram implementados em *Python* com a biblioteca *NetworkX* [Hagberg et al. 2008]. O algoritmo *Push-Relabel* é utilizado para a avaliação do fluxo máximo. O algoritmo *MaxFlowRouting* é executado nos experimentos considerando 3 pares diferentes de pesos para os critérios de Fluxo Máximo (FM) e Caminho Mínimo (CM). Os pares escolhidos foram: FM = 2 e CM = -5, que faz com que o algoritmo produza rotas de tamanhos menores; FM = 5 e CM = -5, que equilibra a influência dos dois critérios; e FM = 5 e CM = -1, que favorece a produção de rotas com maior conectividade.

São reportados resultados da execução dos algoritmos de roteamento em grafos de Mundo Pequeno de Watts-Strogatz [Watts and Strogatz 1998]. Os algoritmos foram também executados em grafos representando topologias reais da Internet. Foram selecionadas topologias importantes dos EUA, da Europa, do Brasil e do Japão. A Internet2

[Internet2 2025] é uma rede com pontos de acesso em diversas localidades dos Estados Unidos da América. A rede Géant [GÉANT 2025] é um *backbone* que conecta diversas redes de instituições educacionais e de pesquisa pelo continente europeu. A rede Ipê da RNP [RNP 2025] conecta diversos órgãos e instituições acadêmicas do Brasil. A rede Wide [WIDE Project 2025] é um *backbone* que conecta diversas empresas, universidades e instituições de pesquisa no Japão.

No experimento, as rotas produzidas pelo *MaxFlowRouting* e pelo algoritmo de Dijkstra são comparadas levando em consideração 3 métricas. A primeira métrica é o tamanho médio das rotas. A segunda métrica é a soma dos graus de todos os vértices da rota. A ideia é que um vértice com um grau maior tem mais possibilidades de possuir uma rota alternativa para o destino. A terceira métrica é o número médio de rotas alternativas disponíveis por vértice de cada rota. Uma rota alternativa é uma rota disjunta da rota original produzida pelo algoritmo de roteamento. A métrica é computada como segue. Inicialmente, as arestas percorridas pela rota original são removidas do grafo. Em seguida, para cada vértice da rota, com exceção da origem e do destino, verifica-se se cada um de seus vizinhos possui um caminho para o destino. Para cada vizinho com caminho funcional para o destino, incrementa-se o contador de rotas alternativas associado ao vértice. Ao final, somam-se todos os contadores e divide-se o resultado pelo tamanho da rota, excluindo a origem e o destino. Assim, obtém-se o número médio de rotas alternativas por vértice da rota.

### 3.1. Grafos de Mundo Pequeno

Os grafos de mundo pequeno de Watts-Strogatz [Watts and Strogatz 1998] são grafos de $N$ vértices, gerados a partir de uma topologia em forma de anel em que cada vértice é conectado a seus $k$ vizinhos mais próximos. Com uma probabilidade $p$, cada aresta $(u, v)$ entre vizinhos $u$ e $v$ é substituída por uma aresta $(u, t)$, em que $t$ é um outro vértice do grafo, selecionado aleatoriamente. No experimento, foram utilizados grafos de tamanho $N = 100, 150$ e $200$, com vértices com $k = 4$ vizinhos e probabilidade $p = 0,4$ das arestas serem substituídas. Foram computadas rotas entre todos os vértices do grafo que não possuem aresta entre si. As métricas foram calculadas considerando casos em que as rotas produzidas por cada algoritmo diferem. Os resultados obtidos com o experimento estão apresentados na Tabela 1.

**Tabela 1. Resultados para grafos de mundo pequeno.**

| $N$ | Algoritmo | Tam. Médio | Soma de Graus | Rotas Alternativas |
|---|---|---|---|---|
| 100 | Dijkstra | 5,17 | 22,61 | 2,48 |
| 100 | *MaxFlowRouting* com 2,-5 | 5,61 | 26,41 | 3,06 |
| 100 | *MaxFlowRouting* com 5,-5 | 6,07 | 28,50 | 2,96 |
| 100 | *MaxFlowRouting* com 5,-1 | 10,76 | 54,91 | 3,37 |
| 150 | Dijkstra | 5,54 | 24,26 | 2,49 |
| 150 | *MaxFlowRouting* com 2,-5 | 5,96 | 27,11 | 2,80 |
| 150 | *MaxFlowRouting* com 5,-5 | 6,47 | 31,77 | 3,22 |
| 150 | *MaxFlowRouting* com 5,-1 | 13,98 | 70,97 | 3,28 |
| 200 | Dijkstra | 5,84 | 25,36 | 2,43 |
| 200 | *MaxFlowRouting* com 2,-5 | 6,16 | 28,57 | 2,91 |
| 200 | *MaxFlowRouting* com 5,-5 | 7,08 | 34,08 | 3,05 |
| 200 | *MaxFlowRouting* com 5,-1 | 15,46 | 74,51 | 2,94 |

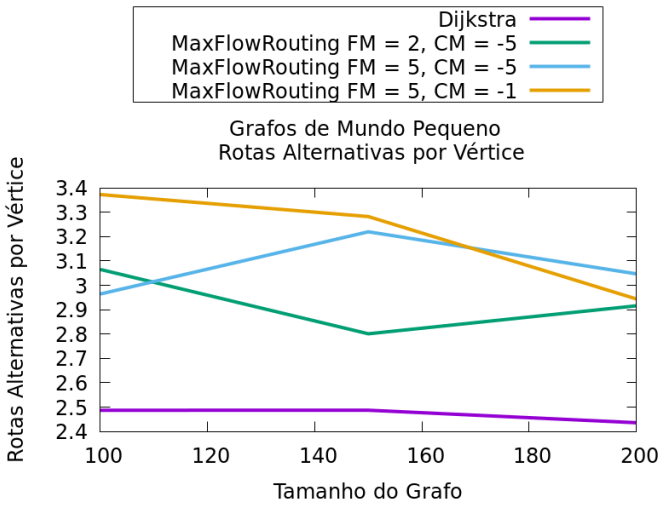


**Figura 4. Comparação da quantidade média de rotas alternativas por vértice. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.**

A Figura 4 mostra que o número médio de rotas alternativas por vértice das rotas produzidas pelo *MaxFlowRouting* ultrapassa o valor obtido pelas rotas computadas pelo algoritmo de Dijkstra. Entretanto, é possível notar uma alta variabilidade nos valores conforme $N$ varia. As rotas computadas pelo algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 apresentaram uma média de 3,37 rotas alternativas por vértice em $N = 100$, caindo para 3,28 em $N = 150$ e ainda mais para 2,94 em $N = 200$.

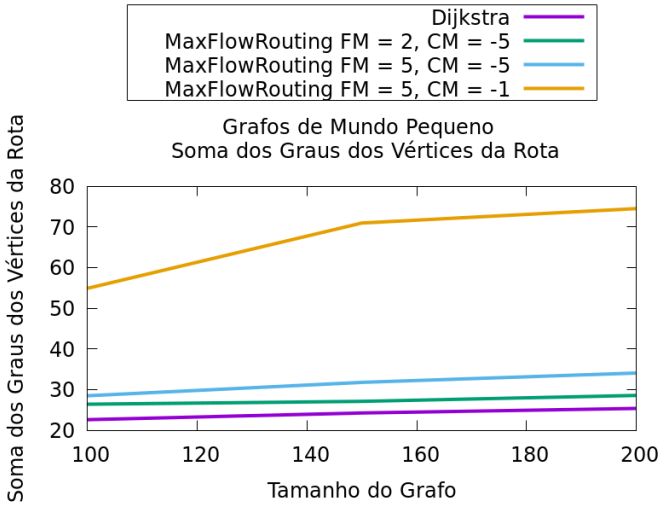


**Figura 5. Comparação da média da soma dos graus dos vértices das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.**

A Figura 5 mostra que a maior média de soma de graus dos vértices ocorre em rotas computadas pelo algoritmo *MaxFlowRouting* com o par de pesos FM = 5 e CM = -1. Isso é esperado, pois esse par de pesos favorece a conectividade nas rotas. Com outros pares de pesos, o *MaxFlowRouting* também mostrou uma vantagem comparado ao algoritmo de Dijkstra, considerando essa métrica.

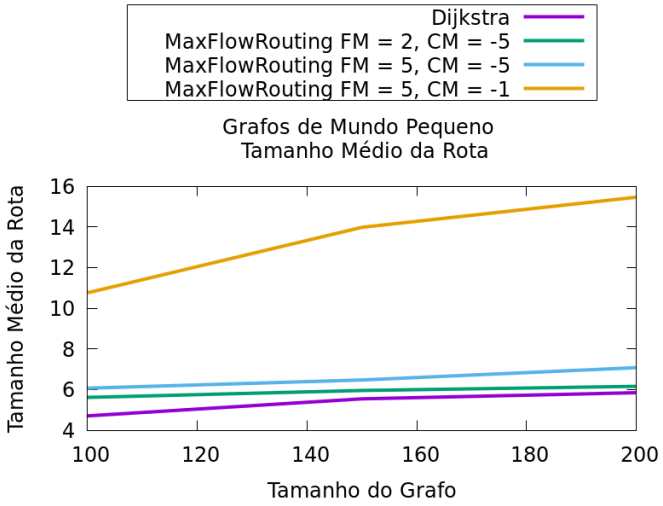


**Figura 6. Comparação do tamanho médio das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.**

A Figura 6 mostra que as rotas computadas pelo *MaxFlowRouting* com pesos FM = 2 e CM = -5 possuem um tamanho médio menor comparadas às rotas produzidas pelo algoritmo com os outros pares de pesos (FM = 5 e CM = -5 e FM = 5 e CM = -1). No melhor caso, as rotas computadas pelo *MaxFlowRouting* foram um pouco maiores do que

**Tabela 2. Resultados do Primeiro Experimento com Grafos Representando Topologias Reais da Internet.**

| Topologia | Algoritmo | Tam. Médio | Soma de Graus | Rotas Alternativas |
|---|---|---|---|---|
| Internet2 | Dijkstra | 9,32 | 24,74 | 0,68 |
| Internet2 | *MaxFlowRouting* com 2,-5 | 9,72 | 26,76 | 0,83 |
| Internet2 | *MaxFlowRouting* com 5,-5 | 9,49 | 26,38 | 0,87 |
| Internet2 | *MaxFlowRouting* com 5,-1 | 10,42 | 28,70 | 0,83 |
| Géant | Dijkstra | 6,37 | 23,94 | 2,14 |
| Géant | *MaxFlowRouting* com 2,-5 | 6,46 | 25,16 | 2,32 |
| Géant | *MaxFlowRouting* com 5,-5 | 6,77 | 26,42 | 2,36 |
| Géant | *MaxFlowRouting* com 5,-1 | 7,61 | 30,11 | 2,37 |
| RNP | Dijkstra | 4,39 | 21,58 | 3,93 |
| RNP | *MaxFlowRouting* com 2,-5 | 4,44 | 25,79 | 5,73 |
| RNP | *MaxFlowRouting* com 5,-5 | 4,42 | 25,54 | 5,71 |
| RNP | *MaxFlowRouting* com 5,-1 | 4,74 | 27,02 | 5,57 |
| WIDE | Dijkstra | 3,44 | 12,22 | 2,00 |
| WIDE | *MaxFlowRouting* com 2,-5 | 3,66 | 17,66 | 5,00 |
| WIDE | *MaxFlowRouting* com 5,-5 | 3,83 | 18,83 | 5,00 |
| WIDE | *MaxFlowRouting* com 5,-1 | 3,83 | 18,83 | 5,00 |

as rotas produzidas pelo algoritmo de FRR baseado em Dijkstra, que computa rotas com o menor tamanho possível.

### 3.2. Topologias Reais da Internet

Os experimentos foram executados em grafos representando as seguintes topologias reais da Internet: Internet2 [Internet2 2025], com $N = 54$ vértices; Géant [GÉANT 2025], com $N = 44$ vértices; RNP [RNP 2025], com $N = 28$ vértices; e Wide [WIDE Project 2025], com $N = 14$ vértices. Foram computadas rotas entre todos os vértices do grafo que não possuem aresta entre si. As métricas foram calculadas somente considerando casos em que as rotas produzidas por cada algoritmo diferem. Os resultados experimentais estão apresentados na Tabela 2.

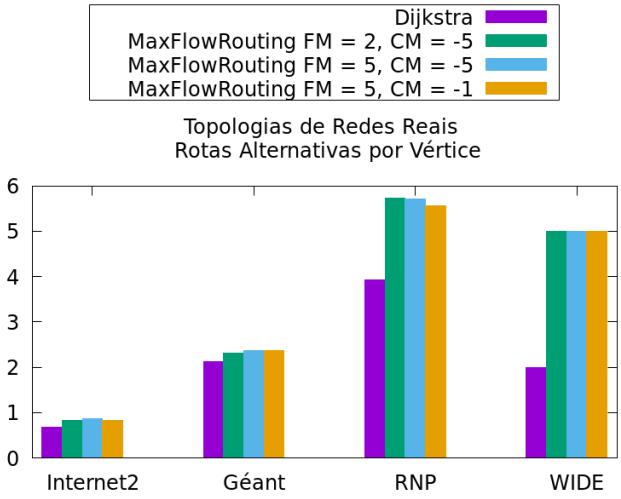


**Figura 7. Comparação da quantidade média de rotas alternativas por vértice. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.**

A Figura 7 mostra que o algoritmo *MaxFlowRouting* produz rotas com mais rotas alternativas por vértice do que o algoritmo de Dijkstra com os 3 pares de pesos. É possível ver também que a variação nos resultados obtidos pelo *MaxFlowRouting* com os diferentes pares de pesos é pequena. A quantidade de rotas alternativas por vértice varia de uma topologia para outra.

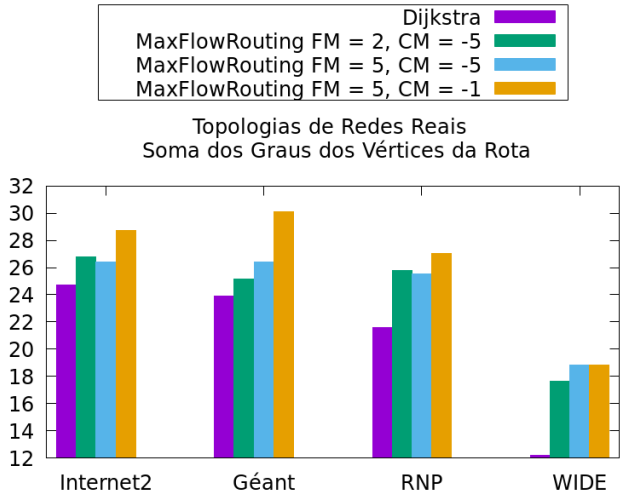


**Figura 8. Comparação da média da soma dos graus dos vértices das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.**

Como apresentado na Figura 8, o algoritmo *MaxFlowRouting* computa rotas com a maior média de soma de graus dos vértices com o par de pesos FM = 5 e CM = -1. O algoritmo *MaxFlowRouting* com outros pares de pesos também ultrapassa o algoritmo de Dijkstra nesse quesito, com destaque para a topologia Wide.

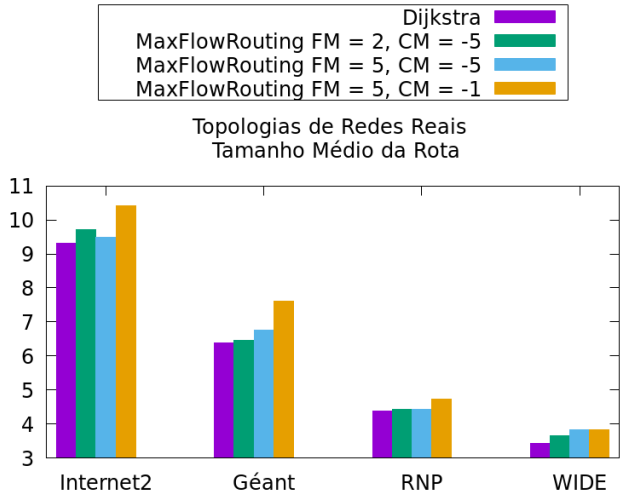


**Figura 9. Comparação do tamanho médio das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.**

A Figura 9 mostra que as rotas computadas pelo algoritmo *MaxFlowRouting* com o par de pesos FM = 2 e CM = -5 são menores quando comparadas às rotas produzidas pelo algoritmo com os outros pares de pesos em quase todas as topologias. As rotas produzidas com esse par de pesos são pouco maiores do que as rotas computadas pelo algoritmo de Dijkstra, que produz as rotas com o menor tamanho possível. Na topologia Internet2, o par de pesos FM = 5 e CM = -5 fez com que o algoritmo tivesse o melhor desempenho nesse quesito comparado aos outros pares de pesos.

## 4. Conclusão

Este trabalho apresentou o algoritmo *MaxFlowRouting* para roteamento tolerante a falhas. O algoritmo apresentado atua no contexto do *Fast-ReRoute*, que é uma estratégia proativa de reparação de falhas, em que uma rota alternativa é pré-calculada e ativada quando falha a rota primária. O *MaxFlowRouting* utiliza a avaliação de fluxo máximo para encontrar rotas com uma quantidade maior de rotas alternativas, além de *backtracking*. Foi apresentada uma avaliação baseada em simulação. Os experimentos foram feitos em grafos de pequeno mundo e em topologias reais da Internet. Os resultados dos experimentos mostram que o *MaxFlowRouting* produz rotas com mais opções de alternativas do que o algoritmo de Dijkstra. No melhor caso, o algoritmo proposto computou rotas com até 1,54 vezes mais rotas alternativas por vértice. Trabalhos futuros incluem a especificação de um protocolo de roteamento TCP/IP baseado no algoritmo *MaxFlowRouting*.

## Referências


Bischof, Z. S., Pitcher, K., Carisimo, E., Meng, A., Bezerra Nunes, R., Padmanabhan, R., Roberts, M. E., Snoeren, A. C., and Dainotti, A. (2023). Destination unreachable: Characterizing internet outages and shutdowns. In *ACM SIGCOMM*, pages 608–621.

Cohen, J., Duarte Jr, E., and Schroeder, J. (2011a). Connectivity criteria for ranking network nodes. In *Complex Networks: Second International Workshop (CompleNet)*, pages 35–45. Springer.

Cohen, J., Rodrigues, L., and Duarte Jr, E. (2012). A parallel implementation of gomory-hu's cut tree algorithm. In *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, pages 124–131. IEEE.

Cohen, J., Rodrigues, L., and Duarte Jr, E. (2017). Parallel cut tree algorithms. *Journal of Parallel and Distributed Computing*, 109:1–14.

Cohen, J., Rodrigues, L., Silva, F., Carmo, R., Guedes, A. L., and Duarte, E. (2011b). Parallel implementations of gusfield's cut tree algorithm. In *The 11th ICA3PP*, pages 258–269. Springer.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.

Duarte, E. and Musicante, M. A. (1999). Formal specification of snmp mib's using action semantics: The routing proxy case study. In *The 6th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 417–430. IEEE.

Duarte Jr, E., Garrett, T., Bona, L., Carmo, R., and Züge, A. (2010). Finding stable cliques of PlanetLab nodes. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 317–322. IEEE.

Duarte Jr, E., Santini, R., and Cohen, J. (2004). Delivering packets during the routing convergence latency interval through highly connected detours. In *International Conference on Dependable Systems and Networks (DSN)*, pages 495–504. IEEE.

Filsfils, C., Francois, P., et al. (2012). Loop-free alternate (LFA) applicability in service provider (SP) networks. RFC 6571.

GÉANT (2025). GÉANT network. `https://network.geant.org/`. Acessado em 28/03/2025.

Hagberg, A., Schult, D., and Swart, P. (2008). Exploring network structure, dynamics, and function using networkx. In *7th Python in Science Conference*, pages 11 – 15.

Internet2 (2025). Layer 1 service. `https://internet2.edu/services/layer-1/`. Acessado em 28/03/2025.

Liu, K., Fan, W., Xiao, F., Mao, H., Huang, H., and Zhao, Y. (2024). Secure paths based trustworthy fault-tolerant routing in data center networks. *Concurrency and Computation: Practice and Experience*, 36(23):e8229.

Maske, C., Cohen, J., and Duarte Jr, E. (2020). Speeding up the gomory-hu parallel cut tree algorithm with efficient graph contractions. *Algorithmica*, 82(6):1601–1615.

Nassu, B., Nanya, T., and Duarte Jr, E. (2007). Topology discovery in dynamic and decentralized networks with mobile agents and swarm intelligence. In *The 7th Int. Conf. Intelligent Systems Design and Applications (ISDA)*, pages 685–690. IEEE.

Okida, L., Maverson, E., and Duarte Jr, E. (2024). Fast reroute with highly connected routes based on maximum flow evaluation. arXiv 2410.10528.

Pan, P. et al. (2005). Fast reroute extensions to RSVP-TE for LSP tunnels. RFC 4090.

RNP (2025). Ipê network. `https://www.rnp.br/en/ipe-network`. Acessado em 28/03/2025.

Schroeder, J. and Duarte Jr, E. (2007). Fault-tolerant dynamic routing based on maximum flow evaluation. In *Latin-American Symposium on Dependable Computing*, pages 7–24. Springer.

Schroeder, J., Guedes, A., and Duarte Jr, E. (2004). Computing the minimum cut and maximum flow of undirected graphs. *Technical report, Federal University of Paraná*.

Shand, M. and Bryant, S. (2010). IP fast reroute framework. RFC 5714.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442.

WIDE Project (2025). WIDE internet official site. `https://two.wide.ad.jp/`. Acessado em 28/03/2025.