João Pedro Amaro da Silva Horta

Licenciado em Engenharia Informática

Encaminhamento com Múltiplas Árvores

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientadores : José Legatheaux Martins, Professor, Universidade Nova de Lisboa
Margarida Mamede, Professora, Universidade Nova de Lisboa

Júri:

Presidente:	Doutor Pedro Manuel Corrêa Calvente Barahona
Arguente:	Doutor Salvador Luis Bettencout Pinto de Abreu
Vogal:	Doutor José Augusto Legatheaux Martins

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

Encaminhamento com Múltiplas Árvores

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Professor José Legatheaux Martins e à Professora Margarida Mamede, pelo apoio constante dado ao longo das várias fases de elaboração deste trabalho. A disponibilidade dos meus orientadores para esclarecer quaisquer dúvidas, o rigor exigido em todos os momentos e as críticas e sugestões recebidas foram essenciais para o desenvolvimento e conclusão deste trabalho.

Aos meus pais e ao meu irmão queria agradecer por sempre me terem apoiado e motivado nesta longa jornada, dando-me sempre força, e por me proporcionarem a possibilidade de completar os meus estudos.

À minha namorada, Inês, pela paciência que sempre teve para ouvir os meus problemas e dúvidas, nunca deixando de me apoiar e motivar, sendo o bastião que me sustentou e sustenta nos momentos mais complicados.

Um muito obrigado também aos meus amigos, especialmente aos "BVA", à "sala 236", à Mariana e à Teresa, que nos momentos em as coisas pareciam complicadas, me fizeram rir e divertir, ajudando a desanuviar e recuperar forças para a tarefa que foi realizar este trabalho.

Por último, queria agradecer aos meus amigos e colegas do Copera, pelo apoio que me deram e pela tolerância que tiveram, quando o desenvolvimento desta dissertação não deixava espaço para poder ajudar na construção do nosso projeto.

Resumo

Nas redes de computadores, o tráfego entre cada par de nós pode ser encaminhado sempre pelo mesmo caminho ou distribuído por vários. Neste trabalho, abordam-se duas facetas do encaminhamento multi-caminho.

Por um lado apresenta-se um novo algoritmo que calcula o conjunto dos caminhos a usar para encaminhar o tráfego entre cada par de nós da rede. A dificuldade do problema advém do facto de que os critérios de seleção dos caminhos podem ser contraditórios. Para privilegiar a comunicação entre o par de nós, devem-se selecionar caminhos de menor custo. No entanto, para aumentar, quer a distribuição de carga entre o par de nós, quer a resistência às falhas, devem-se escolher caminhos disjuntos. O algoritmo proposto tenta conciliar os diferentes requisitos e é parametrizável, para se poder adaptar às diversas características das redes e de forma a controlar a qualidade dos caminhos.

A segunda parte da tese trata o problema da complexidade espacial do encaminhamento multi-caminho dado que o número total de caminhos necessários é potencialmente muito elevado, da ordem de $O(kn^2)$ (sendo k o número de caminhos desejados entre cada par de nós e n o número de nós de entrada ou saída da rede). Reduzir o número de entradas das tabelas de encaminhamento é um objetivo importante, que pode alcançado pela agregação dos caminhos em árvores. Porém, determinar o número mínimo de árvores que cobrem um conjunto de caminhos é um problema NP-difícil. A tese apresenta um novo algoritmo de agregação de caminhos num número reduzido de árvores. A estratégia utilizada privilegia a agregação de caminhos com troços em comum.

Nos testes experimentais efetuados, que envolvem redes sintéticas e reais, ambos os algoritmos produziram melhores resultados que outros previamente publicados.

Palavras-chave: Encaminhamento multi-caminho, seleção de caminhos, problemas de grafos, agregação de caminhos, problemas difíceis

Abstract

In computer networks, the traffic between each pair of nodes can be routed always along the same path or distributed by several. This work addresses two aspects of multi-path routing.

The first part of the thesis presents a new algorithm which computes a set of paths to use for routing traffic between each pair of nodes of the network. The difficulty of the problem comes from the fact that the criteria for selection of paths can be contradictory. The least cost paths should be selected to favor the communication between a pair of nodes. However, to increase both the load distribution between the two nodes and the fault tolerance, one should choose disjoint paths. The proposed algorithm tries to conciliate the different requirements and is parametrizable in order to adapt itself to the various features of the networks and to control the quality of the paths.

The second part addresses the problem of the complexity of multi-path routing, as the total number of required paths is potentially very high, on the order of $O(kn^2)$ (being k the desired number of paths between each pair of nodes and n the number of nodes of origin and destination of traffic in the network). Reducing the number of entries in the routing table is an important goal that can be achieved through aggregation of paths in trees. However, computing the minimum number of trees that covers a set of paths is a NP-hard problem. This dissertation presents also a new algorithm for aggregation of paths in a small number of trees, which uses a strategy that favors aggregation of paths with sections in common.

In the experimental tests conducted, involving synthetic and real networks, both algorithms produced better results than others previously published.

Keywords: Multi-path routing, selection of paths, aggregation of paths, graph problems, hard problems

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

1

Introdução

1.1 Contexto

O crescimento contínuo do número de dispositivos (*hosts*), associado ao aparecimento de um cada vez maior número de aplicações com necessidades específicas de transmissão de dados, tem alterado as características do tráfego na Internet. Devido a esta mudança, o encaminhamento de pacotes nas redes de computadores tem-se alterado e adaptado.

O encaminhamento tradicional, uni-caminho, que consiste na escolha de apenas um caminho entre dois nós, tem tido dificuldade em responder a todos os desafios desafios das redes. Este tipo de encaminhamento é mais simples, sendo mais fácil de gerir e de implementar. No entanto, esta simplicidade dificulta a engenharia de tráfego (*traffic engineering*) e a garantia de qualidade de serviço (*quality of service*). Adicionalmente, falhas que aconteçam na rede têm de ser tratadas com urgência, o que pode introduzir instabilidade e deficiências várias.

Uma alternativa ao encaminhamento tradicional é o encaminhamento multi-caminho, que consiste no uso de vários caminhos para efetuar a comunicação entre dois nós. Apesar de mais complexo e difícil de gerir e implementar, este tipo de encaminhamento permite um melhor aproveitamento dos recursos de uma rede, respondendo melhor aos desafios das redes de maior dimensão.

Em redes de provedores de acesso à Internet (*ISPs - Internet Service Providers*), devido ao grande número de clientes e às necessidades acordadas de qualidade de serviço (*SLAs - Service Level Agreements*), as soluções multi-caminho são comuns. Acontecendo o mesmo nas redes de centros de dados, em que as necessidades de alta capacidade de transmissão obrigam à realização de engenharia de tráfego.

1.2 Motivação

No encaminhamento tradicional, a escolha dos caminhos é normalmente realizada tendo como objetivo a seleção dos mais curtos. Como exemplos do uso deste critério têm-se os protocolos mais conhecidos: *Routing Information Protocol* (*RIP*) [Hed88], *Open Shortest Path First* (*OSPF*) [Moy97] e *Intermediate System to Intermediate System* (*IS-IS*) [Cal90]. A escolha de apenas um caminho entre cada dois nós, associada ao uso do caminho mais curto como o único critério de seleção, facilita o encaminhamento, mas traz algumas limitações. A rede pode ser subaproveitada, correndo o risco de ter canais que são pouco utilizados, enquanto outros são usados por demasiados caminhos. O uso de apenas um caminho por cada par de nós obriga geralmente a que o tratamento de falhas seja baseado em reconfiguração dinâmica da rede. Este processo, com redes de cada vez maior dimensão, pode conduzir a tempos de convergência elevados, que não podem ser aceites pelas garantias de qualidade de serviço exigidas às redes atuais. Adicionalmente, a realização de engenharia de tráfego no encaminhamento uni-caminho apenas é possível em alguns protocolos, com técnicas de alteração de pesos dos caminhos.

O encaminhamento multi-caminho facilita a resposta a estes problemas utilizando vários caminhos para efetuar a comunicação entre cada dois nós da rede. Esta diversidade permite aumentar a taxa de transferência de pacotes, potencialmente tolerar falhas com menos custos para a rede, realizar engenharia de tráfego e aplicar garantias de qualidade de serviço com mais facilidade. O mecanismo mais usado na prática para implementar encaminhamento multi-caminho é o *Multiprotocol Label Switching* (*MPLS*) [RVC01]. Este utiliza circuitos virtuais para permitir o encaminhamento com diferentes caminhos, possibilitando a otimização da rede.

Este tipo de encaminhamento levanta diferentes sub-problemas que se interligam: seleção de caminhos, parametrização dos caminhos nos nós, escolha do próximo nó de destino de um pacote, tolerância a falhas, garantia de qualidade de serviço e engenharia de tráfego. Cada um destes problemas é bastante complexo e tem implicações nas abordagens aos outros.

A escolha de caminhos é a primeira fase do encaminhamento multi-caminho e é fundamental para a qualidade deste. Do conjunto de caminhos escolhidos depende o nível de qualidade de serviço que pode ser fornecido a cada fluxo de tráfego, a capacidade de reação a falhas da rede e a otimização global da capacidade desta. A resolução deste problema pode ser abordada de diversas maneiras, conforme o objetivo que se pretende atingir e quais as informações sobre a rede de que se dispõe. Dados sobre o tráfego e as falhas da rede podem ajudar a uma melhor escolha de caminhos, no entanto, estes nem sempre estão disponíveis, o que limita uma abordagem neles baseada. Adicionalmente, os objetivos da seleção podem ser variados: distribuição de carga, tolerância a falhas, qualidade da comunicação, etc. Estes, muitas vezes são contraditórios, tornando uma solução abrangente e genérica, bastante difícil de implementar. As soluções existentes na literatura para a resolução deste problema são geralmente apresentadas no contexto

de uma solução de encaminhamento particular ([Hec06; Suc+11; SMY00; Mud+10]). A grande maioria destas foca-se apenas num objetivo para a seleção (por exemplo tolerância a falhas ou distribuição de carga).

Esta escolha, por poder ser bastante complexa e pesada, muitas vezes realiza-se *a priori*, sendo só depois os caminhos parametrizados nos nós da rede. No entanto, por ser no contexto do encaminhamento multi-caminho, o número de caminhos selecionados é potencialmente muito elevado, da ordem de $O(kn^2)$ (onde n é a cardinalidade do conjunto de nós de entrada / saída de tráfego e k é o número de caminhos distintos entre cada par desses nós). Assim, a parametrização dos caminhos nos nós pode ser muito complexa e o número de entradas na tabela de encaminhamento bastante elevado. Existe então a necessidade de realizar uma agregação de caminhos, de forma a reduzir a sua complexidade espacial e de parametrização nos nós. Uma solução é a agregação de caminhos em árvores. Estas podem ser parametrizadas em equipamentos normais, através de diversos mecanismos, como MPLS ou VLANS. No entanto, o problema de agregação de caminhos num número mínimo de árvores é NP-difícil [BGN03; Mud+09]. Ao nível da literatura, as soluções existentes para este problema, ou foram feitas para resolver apenas certos sub-problemas [SMY00; BGN03], ou não são deterministas [Mud+09; Mud+10].

1.3 Objetivos

O objetivo deste trabalho é a criação de um solução genérica para a seleção de caminhos e o desenvolvimento de um algoritmo de agregação de caminhos melhor que os publicados até hoje.

O algoritmo de seleção de caminhos deve selecionar um número pedido de caminhos entre dois nós da rede de forma independente da estratégia de encaminhamento e sem depender de informações sobre tráfego ou falhas. Este deve ser parametrizável, de forma a controlar a qualidade dos caminhos e de forma a permitir a adaptação a redes com diferentes características. Adicionalmente, o algoritmo deve selecionar caminhos que permitam maximizar e otimizar o tráfego, suportando um certo número de falhas, sem desprezar o controlo da qualidade dos caminhos.

O algoritmo de agregação deve agregar um conjunto de caminhos num número reduzido de árvores, de forma determinista e em tempo polinomial.

1.4 Principais Contribuições

De seguida são referidas as principais contribuições apresentadas por esta dissertação.

- Desenvolvimento de um algoritmo genérico de seleção de caminhos com base apenas na topologia da rede.
- Criação de um novo algoritmo de agregação de caminhos em árvores.

A análise dos algoritmos desenvolvidos foi realizada por confrontação dos seus resultados com os de outros algoritmos previamente publicados. De forma geral, constatou-se que ambos os algoritmos apresentam globalmente melhores resultados. A análise foi conseguida com base em diversos conjuntos de redes, sintéticas e reais.

1.5 Organização da dissertação

No capítulo 1 fez-se uma pequena introdução sobre o que será tratado nesta dissertação, explicando o contexto em que aparece e quais os principais problemas que a motivam. Adicionalmente apresentou-se uma listagem das principais contribuições deste trabalho.

No capítulo 2 são apresentados os temas que enquadram este trabalho. Como suporte à explicação e enquadramento das várias vertentes, são apresentados os trabalhos já realizados por outros autores. Os temas abordados neste capítulo serão o encaminhamento uni-caminho, o encaminhamento multi-caminho, a seleção de caminhos e a agregação de caminhos.

No capítulo 3 é apresentado o novo algoritmo de seleção de caminhos. De seguida, no capítulo 4 são apresentadas as redes utilizadas nos testes experimentais realizados aos algoritmos. No capítulo 5 são apresentados e analisados os resultados experimentais do algoritmo de seleção. No capítulo 6 é apresentado o novo algoritmo de agregação de caminhos em árvores. No capítulo 7 são apresentados e analisados os resultados experimentais do algoritmo de agregação.

Por último, no capítulo 8, são apresentadas as conclusões finais desta dissertação e são propostos desenvolvimentos futuros no seguimento deste trabalho.

2

Trabalho Relacionado

Neste capítulo ir-se-ão analisar as diversas vertentes que enquadram este trabalho. Cada uma será explicada, mostrando o conhecimento já construído e indicando as semelhanças e as diferenças relevantes das várias abordagens.

Na secção 2.1 falar-se-á do encaminhamento uni-caminho, das suas principais características e das implementações principais existentes. Na secção 2.2 será tratado o encaminhamento multi-caminho, que recorre à utilização de vários caminhos entre dois nós. Na secção 2.3 será analisada a problemática da escolha de caminhos, analisando quais os diferentes critérios de seleção usados em trabalhos anteriores. O foco desta secção será os critérios de seleção, não sendo o cálculo dos caminhos o principal alvo de atenção. Por último, na secção 2.4, serão analisadas as técnicas de agregação de caminhos, em particular através de árvores.

2.1 Encaminhamento uni-caminho

O encaminhamento uni-caminho é um tipo de encaminhamento em que é calculado e usado apenas um caminho entre cada dois nós da rede. O cálculo deste caminho é normalmente realizado usando algoritmos que descobrem o caminho mais curto. Algoritmos baseados no de Dijkstra ou no de Bellman–Ford [Cor+09] são os mais comummente utilizados. Tanto um algoritmo como o outro calculam os caminhos mais curtos entre um nó de um grafo e todos os outros.

2.1.1 Objetivos principais

Este tipo de encaminhamento normalmente pretende garantir: a comunicação através do melhor caminho, a não existência de ciclos (*loops*) na rede e a reação e adaptação do encaminhamento em caso de falhas na rede.

2.1.1.1 Comunicação através do melhor caminho

A comunicação através do melhor caminho tenta que os dados cheguem ao seu destino da forma mais rápida possível. A cada canal da rede é atribuído um peso ou custo. Usando os algoritmos referidos em 2.1, são calculados os caminhos mais curtos com base nos custos que foram atribuídos.

Em teoria, o uso do melhor caminho garantiria a comunicação mais rápida possível entre dois nós. No entanto, usar apenas o melhor caminho pode levar ao subaproveitamento da rede. Pode ocorrer uma concentração excessiva de tráfego numa secção da mesma, derivado de apenas serem escolhidos os melhores caminhos, ignorando outros existentes. Esta concentração pode levar a atrasos e falhas na comunicação, devido a filas nos nós de comutação e o consequente descartar de pacotes.

2.1.1.2 Evitar ciclos na rede

Um ciclo numa rede ocorre quando um pacote enviado a partir de um nó chega de novo a esse mesmo nó, geralmente através de uma interface diferente daquela a partir da qual foi originalmente enviado. Este problema pode acontecer se houver uma falha no algoritmo de cálculo dos caminhos ou se a rede estiver num estado instável devido a falhas. Por exemplo, devido a uma falha num nó da rede, pode acontecer que dois nós da rede pensem que alcançam um terceiro através um do outro. Ou seja, o nó A tenta chegar a C através de B, e o nó B tenta chegar a C através de A, criando um ciclo na rede.

A existência de um ciclo na rede provoca congestionamentos nos nós de comutação envolvidos. No caso de se tratar de uma rede que funcione com base em protocolos de encaminhamento por inundação (*c.f.* secção 2.1.2.1), um ciclo pode causar uma *broadcast storm*: produz-se uma quantidade excessiva de tráfego devido a inundações constantes, algo que pode impedir a conectividade.

Uma técnica usada nas redes de pacotes para evitar que devido a um ciclo um pacote de dados circule pela rede durante tempo indeterminado é o *Time to Leave* (*TTL*). Um pacote *IP* carrega consigo no cabeçalho um número, o *TTL*, que à passagem por cada nó é decrementado. Quando o *TTL* chega a 0 o pacote é descartado, deixando de circular pela rede.

2.1.1.3 Adaptação em caso de falhas

Os nós e os canais da rede podem falhar, tanto a nível de *hardware* como de *software*. Estas falhas deixam a rede num estado inconsistente, podendo invalidar alguns dos caminhos anteriormente calculados, por falha de um dos componentes envolvidos nesses caminhos. Visto apenas existir um caminho calculado entre cada dois nós, em caso de falha na rede, os caminhos devem ser recalculados pelos nós de comutação de forma a tentar garantir a conectividade na rede.

A reconfiguração da rede em caso de falhas obriga a uma troca de mensagens entre os vários nós de comutação. Este processo dura enquanto o algoritmo distribuído de cálculo de caminhos corre nos vários nós, terminando apenas quando a rede se encontrar num estado estável, em que todos os participantes têm dela uma visão correta. Ao tempo decorrido entre a entrada da rede num estado inconsistente e o término da sua reconfiguração chama-se tempo de convergência. Este depende, entre outros fatores, de temporizadores (*timers*) cujo valor influencia o quão rápido a rede converge para um estado estável. A função dos temporizadores é evitar que a rede ou um caminho seja considerado estável e correto antes de tempo ou, por outro lado, impedir que um caminho que está estável seja considerado em falha. Por vezes, em falhas de curta duração (*transient faults*), o tempo de convergência da rede pode ser maior que a duração da própria falha, como explicado em [CCK10]. Em [Fra+05] este problema é abordado, oferecendo uma solução para alcançar convergência na rede no menor tempo possível.

2.1.2 Implementações

A implementação de encaminhamento uni-caminho é feita utilizando diferentes protocolos. Estes podem ser divididos em três tipos principais: inundação, *distance vector* e *link-state*. Adicionalmente, existe uma técnica que pode ser usada em complemento a um protocolo: os circuitos virtuais.

2.1.2.1 Inundação

Através dos algoritmos de inundação, os nós encaminham os pacotes enviando-os para todos os canais, exceto para aquele através do qual receberam o pacote. Estes protocolos são de uma grande simplicidade e garantem que, caso exista um caminho na rede até ao nó de destino, o pacote será entregue.

No entanto, o uso de inundação ocupa largura de banda inutilmente, dado que cada pacote pode ter vários duplicados a percorrer a rede, quando um deles já chegou ao destino pretendido. A criação de ciclos também pode ser um problema, pois alguns pacotes podem circular indefinidamente pela rede. Estes problemas agravam-se com o crescimento do número de nós da rede. A construção da rede com uma topologia em árvore permite evitar ciclos e anular os duplicados, reduzindo o número de pacotes inúteis. Adicionalmente, o uso da técnica de aprendizagem pelo caminho inverso (*e.g. MAC Address*

learning) consegue eliminar por completo os pacotes inúteis na rede. Ao receber um pacote vindo de um nó N, é registado que através dessa interface se chega a esse nó. Desta forma, da próxima vez que chegar um pacote que tem como destino o nó N, o pacote apenas é encaminhado através da interface registada. Esta técnica permite que, ao fim de algum tempo de funcionamento, cada nó já tenha informações sobre atrás de que interfaces estão os outros. Assim, é possível otimizar o tráfego limitando ao máximo a existência de pacotes inúteis.

Nas redes *ethernet* comutadas (*Switched Ethernet*) existe o problema dos duplicados. Este pode ser resolvido se a topologia da rede for uma árvore ou se se usar o protocolo *Spanning Tree Protocol* (*STP*) [Per85]. Este último protocolo cria uma árvore de cobertura na rede, com raiz num nó de comutação escolhido segundo um critério, por exemplo, o menor *MAC Address*. Todo o encaminhamento é realizado por inundação apenas por essa árvore, evitando assim ciclos na rede.

Alguns protocolos de outros tipos usam no seu próprio algoritmo inundação para difundir informações sobre a rede. Por exemplo, o protocolo *Open Shortest Path First* (*OSPF*) [Moy97] usa inundação fiável para difundir os *Link State Anouncements* (*LSAs*), que contêm a informação sobre os canais a que cada nó se encontra ligado. O protocolo tem mecanismos próprios para suprimir os duplicados.

2.1.2.2 Distance Vector

Distance vector é um tipo de protocolo, baseado no algoritmo de *Bellman–Ford*, em que cada nó de comutação calcula, com a informação recebida a partir dos seus vizinhos, o caminho mais curto entre si e todos os outros que conhece. De forma a todos os nós terem uma visão atualizada e consistente da rede, cada um envia aos seus vizinhos a sua visão da rede. Assim, cada nó, após recalcular a sua visão própria da rede, pode encaminhar os pacotes de dados com base nessa visão.

Este tipo de protocolo revela algumas deficiências, como a possibilidade de criação de ciclos temporários na rede. No entanto, apesar de alternativas que resolvem estes problemas já terem sido propostas, as soluções ou são geralmente muito complexas ou necessitam de temporizadores demasiado elevados, criando atrasos na rede em caso de falha, devido a tempos de convergência demasiado grandes.

O exemplo mais simples e conhecido deste tipo de protocolos é o *Routing Information Protocol* (*RIP*) [Hed88]. Neste protocolo um caminho pode no máximo passar por quinze nós. Um caminho de comprimento maior ou igual a dezasseis é considerado com custo infinito, ou seja, inexistente. Alguns dos problemas descritos tentam ser minimizados por este protocolo usando as técnicas *split horizon* e *poison reverse*. O *split horizon* consiste na opção de um nó não anunciar para outro nó os destinos que vê por detrás dele, ou seja, se o nó A chega ao nó C através de B, A não irá anunciar a B que vê C. O *poison reverse* consiste na técnica de anunciar a um nó um destino com custo infinito, se se chega a esse destino através desse nó. Estas duas técnicas tentam evitar que se criem ciclos na

rede. Contudo, não o conseguem garantir totalmente.

Existe uma variante do *distance vector* que se chama *path vector*. Este tipo de protocolo propaga para os vizinhos todo o caminho até cada nó, em vez de uma métrica, como o custo. Desta forma, consegue-se aplicar políticas de encaminhamento específicas após analisar os caminhos recebidos. Adicionalmente, este tipo de protocolo evita ciclos na rede, porque cada nó, se encontrar o seu identificador num caminho recebido, percebe que se formaria um ciclo. O protocolo deste tipo mais conhecido é o *Border Gateway Protocol* (*BGP*) [RL95]. Este é utilizado na comunicação entre sistemas autónomos, permitindo que cada um propague apenas os caminhos que lhe interessam, com os custos que quer.

2.1.2.3 Link-state

Nos protocolos do tipo *link-state*, os nós de comutação da rede partilham entre si o estado dos canais a que estão ligados. Todos os nós ficam com uma visão total da rede, calculando individualmente quais os melhores caminhos entre eles próprios e todos os outros.

Este tipo de protocolo permite que apenas sejam trocadas entre os nós as informações sobre os canais incidentes a cada um, em vez de toda a tabela de encaminhamento como ocorre nos protocolos *distance vector*. Esta característica reduz o tamanho das atualizações enviadas entre os nós, diminuindo portanto a carga sobre a rede. No entanto, apesar de permitir uma melhor configuração e adaptação relativamente aos protocolos *distance vector*, os protocolos *link-state* continuam a necessitar de uma escolha correta de valores para os temporizadores. Essa escolha, se errada, pode levar à não deteção de algumas falhas ou, pelo contrário, ao exagero na sua deteção, causando em ambos os casos atrasos e instabilidade desnecessários na rede.

Existem dois protocolos do tipo *link-state* muito usados: *Open Shortest Path First* (*OSPF*) [Moy97] e *Intermediate System to Intermediate System* (*IS-IS*) [Cal90].

2.1.2.4 Circuitos virtuais

Um circuito virtual é uma ligação virtual pré-estabelecida entre dois nós não necessariamente adjacentes de uma rede de pacotes, que permite a comunicação entre os dois nós como se existisse um canal direto entre os mesmos. Esta técnica é geralmente implementada usando uma forma de encapsulamento suplementar, ao nível ligação de dados ou ao nível rede. As formas mais comuns de utilização de circuitos virtuais no sentido aqui indicado são o encapsulamento de pacotes *IP* em *frames MPLS* e o encapsulamento de pacotes *IP* em *frames Ethernet* de uma VLAN específica, selecionada para forçar a escolha de um caminho determinado numa rede comutada. Também se podem usar túneis *IP* sobre *IP*, encapsulando pacotes *IP* em pacotes *IP*. Com efeito, é possível usar esta técnica com quase todos os tipos de encapsulamento imagináveis.

O uso de circuitos virtuais pode ser motivado para forçar um pacote a seguir um dado caminho específico, o caminho associado ao circuito. Esta utilização também se

pode designar por encaminhamento explícito. Outra motivação para a utilização de circuitos virtuais é a diminuição da complexidade do encaminhamento nos equipamentos da rede. Em geral, o encaminhamento ao nível *IP* requer um algoritmo de determinação da interface associada ao endereço de destino do tipo *Longest Prefix Matching*, enquanto que o encaminhamento com base em etiquetas (*labels*) ou identificadores de circuitos virtuais se resume à consulta de uma tabela usando uma chave única. Os algoritmos de emparelhamento usados neste caso (*e.g.* consulta de uma tabela de dispersão) são de custo constante mesmo que o número de etiquetas seja muito elevado o que simplifica os equipamentos de comutação de pacotes. O custo dos equipamentos de comutação pode diminuir se a computação necessária para cada emparelhamento for menor. Adicionalmente, o facto de se utilizar encaminhamento explícito permite que se faça engenharia de tráfego e que se implementem políticas de qualidade de serviço com mais facilidade.

No entanto, este sistema obriga a que, para cada par de nós que quer comunicar, haja um identificador virtual. Isto leva a que, numa rede com N computadores, possam existir $N\text{x}(N-1)$ identificadores diferentes. Esta questão torna-se um problema se o número de computadores da rede for muito grande, pois obriga a que a tabela dos identificadores também o seja.

Como já se referiu, um mecanismo de circuitos virtuais muito utilizado é o *Multiprotocol Label Switching* (*MPLS*) [RVC01]. Este mecanismo permite atribuir identificadores a ligações virtuais, encaminhando os pacotes encapsulados através da rede usando esses identificadores. Um pacote vindo de um computador, ao chegar a um nó de ingresso (*Label Edge Router – LER*), é encapsulado numa ligação virtual (*Label Switched Path – LSP*). A partir desse momento, o pacote é encaminhado pela rede através de nós de trânsito (*Label Switching Router – LSR*), podendo o seu *LSP* ser trocado entre *LSRs*, para permitir a utilização de um menor número de identificadores.

Apesar do encaminhamento do *MPLS* ser realizado ao nível de ligações virtuais, o cálculo dos caminhos que serão usados como *LSPs* pode ser feito usando um protocolo do tipo *Interior Gateway Protocol* (*IGP*), como o *OSPF*, sobre o qual o mecanismo está implementado. Em alternativa, os *LSPs* podem ser estabelecidos através de parametrização estática dos equipamentos.

O *MPLS* permite que os *LSPs* sejam locais, em vez de globais, de forma a tentar minimizar o problema do número de identificadores de *LSP* necessários. Assim, cada *LSP* apenas é único entre os dois nós que são ligados pelo mesmo canal. Isto permite reduzir o espaço necessário para os identificadores de *LSP* e potencialmente o seu número máximo, mas obriga à modificação do cabeçalho em cada nó onde se troca de identificador. O uso de *LSPs Multipoint-to-point* pode minimizar ainda mais o número de identificadores, como proposto em [BGN02; SMY00; SFM08]. Este tipo de *LSP* funciona como uma árvore com raiz no nó de destino e usa um único identificador para todos os troços comuns até à raiz.

2.2 Encaminhamento multi-caminho

No encaminhamento multi-caminho são utilizados simultaneamente vários caminhos para cada par de nós da rede. Cada pacote de dados é então, enviado através de um caminho considerado mais adequado. Mantem-se geralmente a coerência dentro de cada fluxo de dados, ou seja, pacotes pertencentes ao mesmo fluxo são enviados através do mesmo caminho. Neste contexto entende-se por fluxo de pacotes um conjunto de pacotes com origem e destino (endereços e portas) idênticos e pertencendo ao mesmo protocolo, como por exemplo o conjunto de pacotes associados a uma conexão *TCP*. É importante que os pacotes pertencentes a um mesmo fluxo sejam encaminhados pelo mesmo caminho. Uma divisão entre vários poderia levar a atrasos e chegadas de pacotes fora de ordem, o que para certos tipos de tráfego seria um problema.

2.2.1 Vantagens principais

O encaminhamento multi-caminho pode trazer três vantagens principais em relação ao uni-caminho: otimização da rede, isto é, facilidade de execução de engenharia de tráfego, facilidade de aplicação de garantias de qualidade de serviço e menor custo para a rede na tolerância a falhas.

2.2.1.1 Engenharia de tráfego

A engenharia de tráfego (*traffic engineering*) consiste na otimização da rede, de forma a melhorar o seu desempenho, minimizando excessos de carga em alguns canais, quando alternativas se encontram disponíveis [Hec06]. Trata-se de um problema de otimização cujo enunciado mais simples consiste em distribuir o tráfego de tal forma que a utilização de cada canal seja máxima. Deste modo, tenta-se evitar engarrafamentos e desequilíbrios de carga em certos canais.

A existência de diversos caminhos já calculados para o mesmo par de nós permite distribuir carga de forma mais simples, por comparação com as soluções para protocolos uni-caminho. De facto, a otimização da rede com protocolos uni-caminho implica alterar os pesos dos canais, como proposto em [FT00]. Em [LC02] é defendido que as alterações de peso dos canais são procedimentos mais complicados, porque uma mudança num canal pode afetar toda a rede e de formas muitas vezes imprevisíveis. É apresentado como alternativa o encaminhamento multi-caminho, que permite que a distribuição de carga seja realizada decidindo o que fazer com cada fluxo de dados, não afetando os outros fluxos presentes na rede.

2.2.1.2 Qualidade de serviço

A qualidade de serviço é uma característica de cada fluxo de dados que indica que garantias de encaminhamento deve esse fluxo ter. Estas são geralmente expressas em termos de tempo de trânsito, taxa de transmissão de pacotes e taxa máxima de perca de pacotes.

Fluxos de dados diferentes têm garantias de qualidade de serviço diferentes. Por exemplo, um fluxo de *VoIP* (*Voice over IP*) tem necessidades de tempo de trânsito garantido maiores que as de um fluxo *HTTP* que diz respeito a uma página *web*.

Em [LC02] é defendido que, caso se queira oferecer essa garantia ao nível do encaminhamento num protocolo uni-caminho, é necessário reservar largura de banda para cada fluxo. Isto obrigará a uma maior complexidade nos dispositivos de comutação e nos protocolos de encaminhamento, pois será necessário armazenar informações sobre os fluxos, bem como manter a sua consistência em caso de falha. O encaminhamento multi-caminho permite escolher diferentes caminhos para diferentes qualidades de serviço, evitando possíveis engarrafamentos e atrasos em situações de qualidades de serviço mais exigentes.

2.2.2 Tratamento de falhas

Todas as redes têm de lidar com falhas. Estas acontecem nos nós de comutação e nos canais, tendo causas diversas, desde falhas no *hardware*/*software* até falhas devido a acontecimentos exteriores, como um corte de energia. As falhas afetam a disponibilidade de certos nós e caminhos. A rede deve portanto reagir a estas mudanças, de forma a tentar manter a melhor conetividade e qualidade de comunicação possível. Existem dois tipos de abordagens possíveis a este problema: estática e dinâmica.

Na abordagem estática, a rede já foi configurada com um plano de contingência para o caso de falhas. Durante o cálculo dos caminhos que serão utilizados (cálculo obrigatoriamente estático), são definidas quais as alternativas a estes em caso de falha. Por exemplo, para cada caminho calculado pode ser selecionado um caminho de reserva (*backup*), como realizado em [SMY00]. Esta abordagem permite que, se acontecer uma ou mais falhas, a rede rapidamente volte a um estado estável, bastando seguir o plano já definido. Desta forma, os caminhos onde existem falhas deixam de ser utilizados, sendo substituídos por outros previamente calculados. Contudo, para a criação de um bom plano de contingência é necessário ter um modelo de falhas da rede e disponibilidade extra de capacidade. Um modelo de falhas indica que tipos de falhas acontecem na rede, qual a probabilidade de ocorrência e duração, sendo estes dados apenas indicadores que não dão garantias totais. Um modelo deste tipo muda de rede para rede e nem sempre está disponível, o que obriga quem cria o plano a decidir sem uma base estatística do funcionamento da rede.

Na abordagem dinâmica do tratamento de falhas, quando se deteta uma falha, a rede reconfigura-se automaticamente. Em 2.1.1.3 esta abordagem foi explicada no contexto dos protocolos uni-caminho, sendo o problema da mesma natureza para os multi-caminho. Em [CCK10] é defendido que a abordagem dinâmica traz demasiados problemas para o encaminhamento atual, sendo preferível encaminhamento multi-caminho com tolerância a falhas de forma estática. Uma forma de lidar com as falhas em contexto multi-caminho consiste em ter uma tal diversidade de caminhos que garanta que existem

sempre caminhos disponíveis mesmo em caso de falha de alguns.

2.2.3 Cálculo dos caminhos

Os caminhos que serão utilizados têm de ser calculados, podendo esse cálculo ser feito de forma dinâmica (*online*) ou estática (*offline*).

Os protocolos cuja propagação do estado da rede entre os nós é baseada em *link state* como [Hop00; NST99; SN02] calculam os caminhos dinamicamente mas levantam problemas. Por um lado, enquanto os nós realizam o algoritmo distribuído de cálculo dos caminhos, a rede encontra-se num estado inconsistente, podendo causar atrasos e interrupções no tráfego. Por outro lado, tratando-se de um protocolo multi-caminho, a computação necessária para obter todos os caminhos é mais pesada que o habitual, obrigando a uma maior capacidade de processamento nos dispositivos de comutação. Finalmente, para que a computação resulte num melhor conjunto de caminhos são necessárias informações sobre o tráfego. Estas obrigam a uma permuta adicional de informações entre os nós da rede, o que se pode tornar muito pesado e complexo.

No cálculo estático, os caminhos que serão usados são computados *a priori* e só depois parametrizados nos nós da rede. As redes baseadas em circuitos virtuais [RVC01; SMY00] necessitam de calcular os caminhos estaticamente, pois é complexo mudar os circuitos muito rapidamente. Por ser feita de forma desligada da rede, a computação pode ser muito mais pesada e complexa do que se tivesse de ser feita em cada nó. Desta forma cada nó não necessita de gastar processamento no cálculo dos caminhos. Todavia, a computação completamente estática impede que, em reação a mudanças na topologia, a falhas ou a alterações bruscas de tráfego, a rede possa recalcular caminhos melhores que os inicialmente computados.

Os caminhos podem ser selecionados por diferentes critérios (*c.f.* secção 2.3), que podem ser divididos em duas categorias principais: *Equal-Cost Multipath* (*ECMP*) e multi-caminho generalizado.

No *ECMP* apenas são escolhidos os caminhos suplementares cujo custo é igual ao do caminho mais curto. Apesar do seu uso garantir que o encaminhamento é sempre feito pelos caminhos mais curtos, portanto, mais rápidos, este critério pode ignorar caminhos que, não sendo tão bons, trariam mais diversidade. Os recursos da rede podem ser desta forma subaproveitados, pois a engenharia de tráfego é mais limitada.

O multi-caminho generalizado engloba critérios de seleção em que são escolhidos caminhos de vários tipos, não sendo obrigatoriamente os mais curtos. Potencialmente, esta abordagem permite uma melhor engenharia de tráfego e um melhor aproveitamento dos recursos da rede.

2.2.4 Escolha do nó seguinte

A escolha do nó seguinte para onde um pacote de dados deve ser enviado é crucial no encaminhamento multi-caminho. Esta é efetuada em cada nó pelo qual o pacote passa

até chegar ao destino e pode ser dividida em duas categorias: estática e dinâmica.

Na estática, os nós do interior da rede encaminham pacotes com base em caminhos previamente definidos. Este tipo de escolha geralmente requer alguma forma eficiente de determinar o caminho a partir do cabeçalho, por exemplo, utilização de identificadores de circuitos virtuais. De forma a possibilitar esta eficiência, utiliza-se encaminhamento explícito, em que a decisão de quais os identificadores a atribuir a cada fluxo é tomada no momento em que o pacote entra na rede.

Na dinâmica, cada nó analisa o cabeçalho do pacote e toma a decisão com base em informações sobre a rede. A decisão é tomada pacote a pacote, podendo acontecer que dois pacotes do mesmo fluxo sejam encaminhados por caminhos diferentes, se as condições da rede tiverem mudado entretanto. No encaminhamento uni-caminho, as informações sobre a rede são geralmente relativas a falhas e mudanças de topologia. Contudo, no multi-caminho, a obtenção e atualização dinâmica das informações sobre a rede é mais complexa, implicando também dados sobre carga nos canais e caminhos. A escolha dinâmica obriga a que todos os nós tomem decisões muito informadas, ao invés de simplesmente consultarem uma tabela de mapeamento direto. Neste tipo de escolha, os nós do interior da rede necessitam de mais capacidade de processamento. Os dispositivos serão mais caros que os seus pares da abordagem estática, em que as decisões são tomadas em nós de entrada ou previamente de forma independente. Neste caso, os nós de entrada serão dispositivos mais complexos e dispendiosos. No entanto, os nós do interior da rede, que são em maior número e que geralmente concentram mais tráfego, poderão ser mais simples e baratos.

Independentemente de onde e de que forma, podem ser usados diferentes critérios para a escolha do nó seguinte. Podem-se dividir os critérios segundo o uso ou não de informações sobre a carga dos canais e caminhos. Quando a carga não é tida em conta está-se geralmente perante um algoritmo do tipo aleatório ou *round-robin*. Este último divide o tráfego de forma igual entre os vários caminhos selecionados. Podem ser utilizadas funções de dispersão para dividir o tráfego de forma equilibrada como em [Hop00], mantendo os pacotes pertencentes a um fluxo no mesmo caminho ou, simplesmente, encaminhar pacotes à vez por cada caminho, sem qualquer critério de correlação. Esta abordagem, por não ter em conta as condições da rede no momento, pode conduzir a um subaproveitamento de recursos. Por exemplo, pode ocorrer uma situação em que o tráfego é divido 50/50 entre dois caminhos (ou circuitos virtuais), quando na realidade um deles tem muito mais capacidade que o outro. Quando a carga é um fator importante de decisão, a engenharia de tráfego pode ser realizada com mais sucesso, permitindo um melhor uso e controlo dos recursos da rede. No entanto, torna-se muito complexo difundir as informações sobre a carga dos canais entre os nós da rede. Os algoritmos que permitem esta partilha de informações, como [SN02; NZ01], podem trazer uma sobrecarga de tráfego à rede.

2.2.5 Implementações

A implementação de encaminhamento multi-caminho é realizada através de diferentes protocolos e técnicas.

2.2.5.1 Inundação

Uma técnica que permite encaminhamento multi-caminho por inundação é baseada em *Virtual Local Area Networks* (*VLANs*). As *VLANs* são redes virtuais criadas paralelamente à rede normal, em que cada nó da rede pode pertencer a várias *VLANs* distintas. Cada uma possui a sua *Spanning Tree* própria, correndo o *STP* apenas dentro do seu âmbito. O encaminhamento é realizado fazendo inundação de cada pacote apenas dentro da *VLAN* de destino do pacote, ajudando a limitar o desperdício de largura de banda. Desta forma, variando a *VLAN* através da qual se está a encaminhar o pacote, variam-se os caminhos utilizados. Em [Mud+10] é apresentado um protocolo multi-caminho que usa *VLANs* para oferecer diversos caminhos entre cada par de nós. Os caminhos a utilizar são calculados de forma estática, utilizando um critério de multi-caminho generalizado. Posteriormente são agrupados em *VLANs* e só depois parametrizados nos dispositivos da rede. Os critérios de seleção, bem como os de agregação, serão analisados de forma mais detalhada nas secções 2.3 e 2.4 respetivamente. A cada fluxo de dados é inicialmente atribuída, no nó de entrada da rede, uma *VLAN*. A partir desse momento, todos os pacotes desse fluxo são encaminhados de forma estática através dessa mesma *VLAN*. No caso de falha, mudança da topologia ou motivos relativos à engenharia de tráfego, a *VLAN* atribuída a um fluxo pode ser alterada.

O facto do cálculo de quais as *VLANs* a utilizar ser feito de forma estática, e do mapeamento entre fluxos e estas ser realizado nos nós de entrada da rede traz vantagens, como explicado em 2.2.3 e 2.2.4 respetivamente. No entanto, o facto de um dispositivo suportar um número limitado de *VLANs* (no máximo 4096, muitas vezes menos) pode trazer alguns problemas no caso de redes de maior dimensão.

2.2.5.2 Link-state

Existem várias implementações de encaminhamento multi-caminho que usam as capacidades de disseminação de informação dos protocolos *link-state*. Os caminhos selecionados são em alguns casos do tipo *ECMP* e a decisão do encaminhamento é realizada na totalidade pelos nós do interior da rede.

Em [Hop00] é proposto um algoritmo que usa caminhos do tipo *ECMP* e que faz encaminhamento de forma dinâmica. Este é realizado usando uma política *round-robin*, que usa uma função de dispersão para selecionar o caminho a utilizar para cada fluxo de dados.

Em [NST99] é proposto o algoritmo *Multiple Path Algorithm* (*MPA*), que pode ser implementado nos protocolos *link-state* e que utiliza uma estrutura de dados para guardar diferentes caminhos para cada destino. Os pacotes são encaminhados de forma dinâmica,

utilizando as mensagens *link-state* para atualizar a estrutura de dados. Este algoritmo permite estar menos dependente da convergência da rede, porque em caso de falha, cada nó pode encaminhar pacotes através de outros caminhos que não tenham sido afetados. No entanto, o facto de ser necessário guardar uma estrutura adicional nos dispositivos pode trazer um maior custo em termos de armazenamento.

Em [SN02] é proposto o *OSPF Optimized Multipath* (*OSPF-OMP*), um algoritmo para *OSPF* que usa caminhos do tipo *ECMP* de forma a permitir encaminhamento multi-caminho de forma dinâmica. Neste algoritmo, os nós de comutação da rede disseminam informação sobre a carga dos canais por inundação na rede. Desta forma, cada nó toma decisões com base em informações sobre a carga dos canais. O problema do excesso de tráfego produzido pelas mensagens de informação de carga é tratado com alterações nas configurações, por exemplo, modificando os valores dos temporizadores. Os valores dos temporizadores devem ser escolhidos com cuidado, tal como a periodicidade e os critérios das reações que provocam a disseminação das informações.

Em [NZ01] é proposto um algoritmo para protocolos do tipo *link-state*, em que são selecionados caminhos dinamicamente, com base nas informações globais da rede, transmitidas, por exemplo, através dos *LSAs* do *OSPF*. As decisões de encaminhamento são dinâmicas e dependentes das informações sobre a carga nos canais a nível local. Tal como no *OSPF-OMP*, permite a tomada de decisões informadas. Todavia, a configuração híbrida do algoritmo faz com que não sejam necessárias atualizações constantes por inundação.

2.2.5.3 Circuitos virtuais

O *MPLS* permite a configuração estática de diversos caminhos entre cada par de nós e é a solução mais frequentemente utilizada na prática. Na implementação mais comum, os nós da periferia da rede executam as seleções de caminhos que os pacotes devem seguir até ao destino (selecionando o identificador de *LSPs* respetivo) através de critérios de distribuição de carga ou de qualidade de serviço, geralmente estabelecidos estaticamente. Desta forma, é possível atribuir a pacotes, cuja origem e destino são os mesmos, diferentes identificadores e portanto, diferentes caminhos. A seleção destes utiliza um critério do tipo multi-caminho generalizado. Em [SMY00] é proposta uma técnica para criar e utilizar múltiplos *LSPs Multipoint-to-Point* (*m-t-p*), de forma a realizar engenharia de tráfego. Os *LSPs* originais são calculados *a priori*, tal como a sua agregação em *m-t-p*.

2.3 Seleção de caminhos

O encaminhamento multi-caminho traz muitos desafios, sendo o primeiro a seleção de caminhos. Como explicado em 2.2.3, esta pode ser realizada estaticamente ou dinamicamente e os critérios de seleção podem ser do tipo ECMP ou multi-caminho generalizado. Será detalhada nesta secção a seleção estática de caminhos utilizando critérios do tipo

multi-caminho generalizado.

2.3.1 Modelação de redes de computadores através de grafos

A topologia de uma rede de computadores pode ser modelada por um grafo. Este consiste num conjunto de vértices ligados entre si por arestas ou arcos. Na representação de uma rede, os nós da rede são vértices e os canais são arcos. Assim, um grafo que modela uma rede pode ser representado por $G = (V, E)$, em que V é o conjunto dos nós e E o conjunto dos canais. Se todos os canais forem bidirecionais, ou seja, se suportarem tráfego nos dois sentidos, o grafo é não orientado. Se os canais forem unidirecionais, ou seja, apenas suportarem tráfego num sentido, o grafo é orientado, tendo cada arco um sentido único. O grafo da rede geralmente é pesado, isto é, os arcos têm um peso ou um custo associado, que toma um valor positivo. O peso geralmente representa a capacidade de transmissão de um canal. Também é habitual considerar-se que o grafo é simples. Ou seja, não há arcos paralelos (arcos diferentes com a mesma origem e o mesmo destino) nem lacetes (arcos cujas extremidades são o mesmo nó).

Um caminho entre dois nós x e y da rede é uma sequência não vazia de nós tal que, para cada um, à exceção do último, existe um canal que o liga ao nó seguinte da sequência, sendo o nó inicial desta x e o final y. Um caminho simples é um caminho cujos nós são todos diferentes, exceto possivelmente o primeiro e o último. Um caminho com pelo menos dois nós também pode ser representado por uma sequência de canais tal que, para cada dois canais consecutivos (x, y) e (w, z), $y = w$. O peso ou custo de um caminho é a soma dos pesos de todos os canais do caminho. O comprimento de um caminho é o número de canais do caminho.

Um ciclo num grafo orientado consiste num caminho de comprimento positivo onde o primeiro e o último vértices são iguais. Num grafo não orientado, um ciclo é um caminho de comprimento positivo onde o primeiro e o último vértices são iguais e que não passa duas vezes pelo mesmo canal. Um grafo diz-se cíclico se tiver algum ciclo e diz-se acíclico se não tiver quaisquer ciclos.

Um caminho é disjunto de outro em termos de canais se não existirem canais em comum entre os dois. Um caminho é disjunto de outro em termos de nós se não existirem nós em comum entre os dois, com exceção do inicial e final. Esta última disjunção garante também a disjunção em termos de canais. Se um canal faz a ligação entre dois nós, então o uso do canal obriga ao uso desses dois nós. Desta forma, se um nó não é usado, então nenhum dos seus canais é usado.

Uma árvore é um grafo não orientado, conexo (existe um caminho entre cada par de nós) e acíclico. Uma floresta é um grafo não orientado, acíclico e não conexo. Um sub-grafo $G' = (V', E')$ de um grafo $G = (V, E)$ é um grafo tal que $V' \subseteq V$ e $E' \subseteq E$.

2.3.2 Implementações da seleção de caminhos

A seleção de caminhos realiza-se com base na topologia da rede. No entanto, é possível utilizar dois tipos de informações adicionais na seleção: informações sobre as falhas da rede e informações sobre o tráfego da rede.

As falhas são uma preocupação essencial do encaminhamento. O multi-caminho permite que, usando um conjunto diverso de caminhos, se suportem falhas de forma eficiente. Em caso de falha num dos componentes de um caminho, basta redirecionar o tráfego para outro que não tenha sido afetado, não obrigando a rede a reconfigurar-se dinamicamente. Esta capacidade de tolerar falhas é conseguida se o conjunto de caminhos escolhidos o possibilitar. Por exemplo, se para cada falha possível existir uma alternativa já prevista.

Para selecionar os caminhos de forma a suportarem as falhas da rede, é conveniente saber que falhas e combinações de falhas se tem de suportar. Na seleção de caminhos pode ser utilizado um modelo de falhas, de forma a ajudar a escolher o conjunto de caminhos que melhor tolera as falhas na rede. Contudo, pode ser complicado e nem sempre é possível criar um modelo de falhas. Muitos fatores influenciam a ocorrência de falhas e nem todos podem ser medidos ou controlados. Por exemplo, a idade dos dispositivos ou os cortes de energia provocados por eventos inesperados são fatores que introduzem um grau elevado de imprevisibilidade na definição de um modelo de falhas.

Um dos outros objetivos do encaminhamento multi-caminho é a realização de engenharia de tráfego. Esta consegue-se mais facilmente se se souber *a priori* como se comporta o tráfego na rede. Uma matriz de tráfego é uma matriz que representa a quantidade média de tráfego dos fluxos entre cada dois nós da rede. A seleção de caminhos, como veremos a seguir, pode ser realizada utilizando uma matriz de tráfego.

2.3.2.1 Seleção de caminhos usando uma matriz de tráfego

Na seleção de caminhos pode ser tida em conta uma matriz de tráfego. Usando algoritmos de otimização, consegue-se selecionar um conjunto de caminhos ótimo, relativamente ao tráfego previsto para a rede. No entanto, o tráfego numa rede é geralmente imprevisível. Isto deve-se à existência de mudanças bruscas de tráfego e ao dinamismo natural do protocolo *TCP/IP*, que tenta realizar uma otimização da rede, variando a capacidade de comunicação requerida. Não é possível portanto garantir que a rede se comportará exatamente como descrito na matriz de tráfego. Desta forma, um conjunto ótimo de caminhos para um certo tráfego previsto pode não ser tão bom para um tráfego real distinto. Apesar das limitações, a seleção de caminhos com base em previsões de tráfego é utilizada. Por exemplo, [Lee+02; Suc+11] apresentam propostas em que os caminhos são selecionados com base numa matriz de tráfego. No segundo, a seleção é também baseada em informações sobre as falhas da rede.

2.3.2.2 Seleção de caminhos sem matriz de tráfego ou modelo de falhas

Quando não se tem acesso ou não se quer usar uma matriz de tráfego ou um modelo de falhas da rede, a seleção de caminhos é realizada com base em critérios pré-definidos. Este tipo de seleção tem a vantagem de poder ser aplicado a qualquer rede, pois não necessita de qualquer matriz ou modelo. No entanto, uma escolha apenas com base na topologia da rede e em critérios pré-definidos pode conduzir a um conjunto de caminhos que não responde da forma mais eficiente ao comportamento real da rede.

A escolha de quais os critérios a utilizar depende de quais os objetivos a que se pretende dar mais prioridade. Existem quatro critérios tipicamente mais usados: menor comprimento dos caminhos, menor custo dos caminhos, disjunção em termos de canais e disjunção em termos de nós. Serão de seguida avaliadas as vantagens e desvantagens de cada critério.

- Menor comprimento dos caminhos — garante que os caminhos escolhidos atravessam a menor quantidade de nós e canais possível, diminuindo assim a probabilidade de uma falha os inutilizar e aumentando globalmente a sua qualidade. Contudo, traz uma diversidade de caminhos limitada, pois não tem em conta caminhos que, não sendo os menores, permitem aproveitar melhor os recursos da rede. Adicionalmente, estes caminhos não dão nenhumas garantias em termos de falhas, pois podem ter nós e canais em comum. Por último, os canais podem ter custos diferentes. Assim, os menores caminhos podem não ser os melhores caminhos em termos de capacidade de transmissão.

- Menor custo dos caminhos — permite que sejam escolhidos os caminhos com maior capacidade de transmissão, ou seja, potencialmente mais rápidos. No entanto, tal como no critério do menor comprimento, este produz uma diversidade limitada e não dá garantias em termos de falhas.

- Disjunção em termos de canais — permite suportar garantidamente um certo número de falhas de canais. Dado cada caminho não partilhar nenhum canal com outro, se acontecer uma só falha, apenas um caminho é afetado. O número de falhas simultâneas suportadas depende do número de caminhos escolhidos. Se este for k, só são garantidamente suportadas $k - 1$ falhas simultâneas de canais. Apesar das garantias em termos de falhas oferecidas por este critério, a sua aplicação muitas vezes é complicada. A topologia da rede pode não permitir o número de caminhos disjuntos em termos de canais desejado.

- Disjunção em termos de nós — permite suportar garantidamente um certo número de falhas de nós e canais. As garantias oferecidas são semelhantes às da disjunção em termos de canais, tolerando adicionalmente falhas de nós. No entanto, este critério tem também os mesmos problemas que a disjunção em termos de canais, agravando-os. Isto fica-se a dever ao facto da diversidade de caminhos disjuntos em termos de nós ser menor que em termos de canais.

Listing 2.1: Algoritmo de seleção de caminhos proposto em [Mud+10]

```
1
2  Sejam:
3  G = (V, E) - Grafo da rede
4  s - Nó de origem
5  d - Nó de destino
6  k - Número de caminhos desejados entre s e d
7
8  Algoritmo ComputePaths(G, s, d, k)
9  for e ∈ E do
10     w(e) = 1;
11 end
12
13 P = φ;
14
15 while |P| < k do
16     p = shorthestPath(G, s, d, w);
17     if p ∈ P
18         break;
19     else
20         P = P ∪ {p};
21         for e ∈ p do
22             w(e) += |E|;
23         end
24     end
25 end
26
27 return P;
```

Os critérios de disjunção, tanto em termos de nós, como em termos de canais, podem ser relaxados. Ao invés de apenas se aceitar caminhos disjuntos, podem ser selecionados caminhos o mais disjuntos possível. Isto é, minimizando o número de nós ou canais partilhados pelos caminhos. Esta relaxação permite a seleção de mais caminhos do que os critérios originais. No entanto, não dá garantias diretas de tolerância a falhas como os originais.

Estes dois critérios, por não terem em conta o comprimento nem o custo dos caminhos, podem selecionar caminhos muito mais extensos e de maior custo que os caminhos menores e de menor custo. Devido a este risco, são geralmente impostos limites ao comprimento ou ao custo dos caminhos selecionados.

Alguns autores definiram conjuntos de caminhos a selecionar, combinando e adaptando os critérios referidos.

Em [Mud+10] é proposto o algoritmo apresentado na listagem 2.1. Para simplificar, os autores assumem que os arcos do grafo têm todos peso 1. O algoritmo seleciona um caminho de menor custo através da função *shorthestPath* e adiciona ao peso de todos os seus arcos o número de arcos do grafo. Enquanto o número de caminhos desejado não for atingido e o caminho selecionado não existir já no conjunto dos escolhidos, o algoritmo repete; senão, termina e devolve o conjunto dos caminhos selecionados.

Neste caso o primeiro caminho que é calculado é um de menor comprimento. No

entanto, o algoritmo pode ser generalizado para um grafo pesado, alterando o valor que se soma ao peso dos canais de cada caminho selecionado (ver linha 22). Este algoritmo garante que um dos melhores caminhos pertence ao conjunto dos selecionados. Adicionalmente, privilegia a disjunção de canais entre os caminhos do grupo e garante que, entre caminhos que providenciam o mesmo grau de disjunção, é selecionado um de menor comprimento. Contudo, são ignoradas as falhas de nós.

Em [SMY00] o conjunto dos caminhos selecionados entre cada dois nós satisfaz as seguintes restrições.

1. Cada caminho selecionado tem de ter pelo menos outro no conjunto com o qual é disjunto em termos de nós.

2. O comprimento de cada caminho selecionado tem de ser menor que o dos caminhos mais curtos entre os nós mais h, que é um parâmetro dado.

Se não existir nenhum par de caminhos que cumpra estas condições, é escolhido um par disjunto em termos de nós, com o menor comprimento de cada um dos caminhos possível.

O primeiro e o segundo critério conseguem garantir tolerância a uma falha em termos de nós e canais. Esta seleção de caminhos permite que se atribua com facilidade a cada caminho um que pode servir de reserva, ou com o qual pode ser partilhado tráfego. Adicionalmente, é garantido que o comprimento do caminho nunca é maior que um certo limite, o que elimina caminhos que podem ser demasiado longos. No entanto, a solução de recurso para o caso de não existir nenhum par de caminhos que cumpra os critérios iniciais pode selecionar caminhos de fraca qualidade. O facto de não serem impostos limites ao comprimento dos caminhos pode levar à escolha de caminhos demasiado grandes, ao ponto de prejudicarem o tempo de transmissão dos pacotes de dados.

2.4 Agregação de caminhos

O encaminhamento multi-caminho permite potencialmente uma melhor engenharia de tráfego, um melhor cumprimento de garantias de qualidade de serviço e uma melhor tolerância a falhas quanto maior for o número de caminhos que tiver à sua disposição. O número de caminhos selecionados cresce com o aumento do tamanho da rede. Por exemplo, se a rede tiver 300 nós e se se pretender garantir pelo menos 3 caminhos diferentes entre cada dois nós, o número total de caminhos será cerca de 270000. Assim, quanto mais nós a rede tiver, maior será o número de caminhos escolhidos e maior a complexidade espacial e de parametrização em cada nó de comutação. Uma solução para este problema é a agregação de caminhos. Esta consiste na junção de um conjunto de caminhos num certo número de sub-grafos da rede. Assim, pretende-se permitir o encaminhamento pelos caminhos selecionados, mas armazenando apenas informações sobre os sub-grafos, em vez de sobre os caminhos. Como o objetivo é reduzir a complexidade espacial das

estruturas de dados necessárias para codificar os caminhos, geralmente pretende-se o número mínimo de sub-grafos. O tipo mais utilizado de sub-grafo na agregação de caminhos é a árvore, existindo no entanto uma proposta que utiliza grafos orientados acíclicos [SFM08].

2.4.1 Agregação em árvores

A utilização de uma árvore permite que o encaminhamento se faça sem risco de criação de ciclos. Adicionalmente, encaminhamento com endereços hierárquicos e encaminhamento *multicast* são mais facilmente implementados numa topologia em árvore. Contudo, o problema da agregação de caminhos num número mínimo de árvores é NP-difícil, como é provado em [Mud+09; BGN02].

Em [BGN02] é proposto um algoritmo de agregação para *MPLS* (ver listagem 2.2) que junta *LSPs point-to-point*, que representam caminhos entre dois nós da rede, em *LSPs multipoint-to-point*. Este processo pode ser equiparado à agregação de caminhos com o mesmo nó de destino em árvores.

Listing 2.2: Algoritmo de agregação de caminhos proposto em [BGN02]

```
1
2  Seja:
3  P - Conjunto de caminhos com o mesmo nó de destino
4
5  Algoritmo MergingAlgorithm(P)
6  compute all merging indices of the paths P;
7  while exists m_ij ≠ 0 do
8      choose i and j with maximum m_ij;
9      update indices of all trees with respect to i, j
10     merge i and j
11 end
```

Listing 2.3: Algoritmo de criação de índices de fusão [BGN02]

```
1
2  Seja:
3  P - Conjunto de caminhos com o mesmo nó de destino
4
5  Algoritmo ComputeMergingIndices(P)
6  e = target node of the paths in P
7  for each distinct pair p_i , p_j ∈ P do
8      Chain = {e};
9      while previousNode_i == previousNode_j do
10         Chain = Chain ∪ {previousNode_i};
11     end
12     remove nodes in Chain from p_i and p_j;
13     if there are common nodes in p_i and p_j
14         m_ij = 0;
15     else
16         m_ij = |Chain|;
17     end
18 end
```

Listing 2.4: Algoritmo de atualização de índices de fusão [BGN02]

```
1
2  Sejam:
3  m_ki - Índices de fusão das árvores k e i
4  m_kj - Índices de fusão das árvores k e j
5  m_ij - Índices de fusão das árvores i e j
6
7  Algoritmo UpdateMergingIndices(m_ki, m_kj, m_ij)
8  if m_ki == 0 || m_kj == 0
9      m_k(ij) = 0;
10 else
11     m_k(ij) = m_ki;
12 end
```

Inicialmente, o algoritmo calcula o índice de fusão (*merging index*) para cada par de caminhos (ver listagem 2.3). Seja k o comprimento do maior sufixo comum dos caminhos (vistos como sequências de nós). Se os nós partilhados entre os caminhos forem apenas os do maior sufixo comum, o índice de fusão é k; senão, é zero. Note-se que a agregação de caminhos cujo índice de fusão é positivo resulta num sub-grafo acíclico.

A segunda parte do algoritmo utiliza os índices de fusão para efetuar a agregação dos caminhos. Inicialmente, é criada uma árvore com cada caminho. De seguida, é selecionado um par de árvores com o maior índice de fusão. Os índices de fusão que envolvem qualquer uma das árvores selecionadas são então atualizados (ver listagem 2.4). Por último, é feita a fusão dessas árvores. Este processo repete-se até não existirem índices de fusão positivos. A complexidade temporal deste algoritmo é $O(n_e^2 l_e)$ por cada nó e da periferia, em que n_e é o número de caminhos selecionados cujo destino é e e l_e o comprimento máximo desses caminhos [BGN02].

Em [SMY00] os caminhos são também compactados em *LSPs multipoint-to-point* acíclicos, sendo este problema formulado como um problema de programação linear inteira do tipo 0-1. O programa é executado para cada grupo de caminhos com o mesmo nó de destino. De seguida é apresentada a formulação do problema.

Conjuntos:

- $\{e\}$ - Conjunto com o nó de destino.
- N - Conjunto de nós.
- I - Conjunto de nós de origem dos caminhos com o nó de destino e.
- L - Conjunto dos canais. Este conjunto é gerado a partir do conjunto de todos os canais, retirando aqueles que têm um nó de I como destino e aqueles que têm o nó e como origem.
- P_i - Conjunto de caminhos com origem no nó i e destino no nó e.
- L^p - Conjunto de canais do caminho p.
- T - Conjunto de árvores candidatas.

Variáveis:

- r^t - Toma valor 1 se a árvore candidata t incluir uma parte dos caminhos selecionados; senão, toma valor 0.
- $h^t_{l,m}$ - Toma valor 1 se a árvore candidata t tiver o canal (l, m); senão, toma valor 0.
- δ^t_p - Toma valor 1 se a árvore candidata t incluir o caminho p; senão, toma valor 0.

Função objetivo (que minimiza o número de árvores):

$$\text{Min} \sum_{t \in T} r^t$$

Restrições:

- $\sum_{\{m|(l,m)\in L\}} h^t_{l,m} \leq 1$ para $\forall_{t\in T}$ e $\forall_{l\in N\setminus\{e\}}$
  Em cada árvore t, para cada nó l (exceto a raiz), o número de arcos que saem do nó é ≤ 1.
- $\sum_{\{(l,m)\in L^p \cap L\}} h^t_{l,m} \geq |L^p| \delta^t_{p_i}$ para $t \in T$, $i \in I$ e $p \in P_i$
  Se uma árvore incluir um caminho, todos os canais do caminho pertencem à árvore.
- $\sum_{t\in T} \delta^t_p \geq 1$ para $\forall_{i\in I}$ e $\forall_{p\in P_i}$
  Qualquer caminho está incluído em alguma árvore.

- $\sum_{i \in I} \sum_{p \in P_i} \delta_p^t \leq \sum_{i \in I} |P_i| r^t$ para $\forall_{t \in T}$

  Se uma árvore t incluir um ou mais caminhos, r^t toma valor 1.

- $h_{l,m}^t, \delta_p^t, r^t = 0/1$

  As variáveis podem tomar apenas os valores 0 ou 1.

Aplicando a estratégia utilizada nestes dois últimos algoritmos ([BGN02][SMY00]) de só agregar na mesma árvore caminhos com o mesmo nó de destino ao problema mais geral de agregação de caminhos (bidirecionais) em árvores, o número total de árvores resultante nunca seria muito reduzido. Isto pode ser comprovado com o seguinte raciocínio: dada uma rede com n nós de origem e de destino de tráfego, em geral, um conjunto S de caminhos na rede tem k caminhos para cada par (x, y) desses nós, com $x < y$. Ou seja, $|S| \approx k\frac{n(n-1)}{2}$. Particionando S pelo vértice final dos caminhos, seriam gerados $n-1$ problemas de agregação com o mesmo destino e, para cada um deles, o conjunto de árvores retornado teria, no mínimo, k árvores, porque cada um dos k caminhos com a mesma origem (e destino) teria de ser coberto por uma árvore diferente. Portanto, obter-se-iam pelo menos $(n-1)k$ árvores no total, não sendo esperadas repetições.

Em [Mud+10] é proposto um algoritmo de agregação (ver listagem 2.5) que agrupa os caminhos em sub-grafos acíclicos não necessariamente conexos. Estes sub-grafos podem ser árvores ou florestas.

Este algoritmo funciona de forma integrada com o algoritmo 2.1. Para cada par de nós é calculado o conjunto de caminhos a utilizar. Esse conjunto é então percorrido, escolhendo um caminho de cada vez por ordem aleatória. Ao ser selecionado, tenta-se adicionar o caminho a um sub-grafo já existente, sendo a ordem de escolha dos sub-grafos também aleatória. Se não for possível adicionar o caminho a nenhum sub-grafo de forma a que não crie ciclos, é criada um novo sub-grafo com o caminho.

Este algoritmo não oferece garantias de convergência para o resultado ótimo, sendo necessário executá-lo várias vezes, escolhendo o conjunto resultado com menor número de sub-grafos. Adicionalmente, os autores admitem que este algoritmo não é escalável, sendo a sua complexidade temporal $O(mkn^3)$, em que m é o número de sub-grafos, k o número de caminhos entre dois nós e n o número de nós da rede [Mud+10].

Devido à fraca escalabilidade, em [Mud+09], os autores propõem uma solução alternativa para a agregação de caminhos, que é composta por duas fases distintas. Na primeira, o problema da minimização de sub-grafos acíclicos é paralelizado, utilizando um algoritmo (ver listagem 2.6) que calcula um conjunto reduzido de sub-grafos para cada conjunto de caminhos com o mesmo nó de destino. Este problema é reduzido ao problema conhecido de coloração de vértices. Após calcular o conjunto de caminhos com o mesmo destino, para cada caminho do conjunto é criado um vértice no grafo a colorir. Para cada par de caminhos cuja agregação não é compatível numa árvore ou floresta, por dar origem a um sub-grafo cíclico (utilizando a função *SubGraphCompatible*, ver linha 18),

é criado um arco entre os respetivos vértices. É utilizada então uma heurística [JT92][1] para resolver o problema da coloração de vértices, que retorna o número de cores necessárias e o mapeamento entre vértices e cores (ver linha 23). Por último, os caminhos cujos vértices foram coloridos com a mesma cor são agrupados no mesmo sub-grafo (utilizando a função *MergePaths*, linha 27), que é um sub-grafo acíclico.

Listing 2.5: Algoritmo de agregação de caminhos proposto em [Mud+10]

```
1
2  Sejam:
3  G = (V, E) - Grafo da rede
4  k - Número de caminhos desejados ente cada dois nós da rede
5
6  Algoritmo SubGraphPackingHeuristic(G, k)
7  SG = φ;
8
9  for s ∈ V do
10     for d ∈ V do
11         P = ComputePaths(G, s, d, k);
12         for p ∈ P do //a permutação de P é aleatória
13             if p not covered by any graph in SG
14                 Success = false;
15                 for S ∈ SG do //a permutação de SG é aleatória
16                     if Success == false and p does not create a loop in S
17                         add p to S;
18                         Success = true;
19                     end
20                 end
21                 if Sucess == false
22                     a = new graph with p;
23                     SG = SG ∪ {a};
24                 end
25             end
26         end
27     end
28 end
29
30 return SG;
```

O número de sub-grafos resultantes da execução deste algoritmo para todos os destinos é definido por $\sum_{d \in D} |SG_d|$, em que D é o conjunto dos destinos e SG_d é o conjunto dos sub-grafos calculados para o destino d. Se o número de destinos na rede for elevado, facilmente o número de sub-grafos será muito elevado.

A segunda fase pretende realizar a redução do número de sub-grafos. Nesta é utilizado um algoritmo (ver listagem 2.7) que faz a agregação dos sub-grafos obtidos no passo anterior. Neste algoritmo, o conjunto de todos os sub-grafos calculados para os diferentes destinos é percorrido, escolhendo um sub-grafo de cada vez por ordem aleatória.

[1] Esta é a referência citada no artigo [Mud+09], não tendo sido o original lido no âmbito desta dissertação.

Ao selecionar um sub-grafo, este é retirado do conjunto dos sub-grafos. De seguida são percorridos os outros sub-grafos do conjunto, sendo testado se a sua junção ao sub-grafo selecionado inicialmente cria um grafo acíclico (função *IsThereALoop*, ver linha 12). Se criar um grafo acíclico, o segundo sub-grafo é agregado ao primeiro. Desta forma, para cada sub-grafo do conjunto, é testada a agregação com todos os outros que ainda não foram agregados e portanto retirados do conjunto. O resultado final é um conjunto de sub-grafos acíclicos resultantes da agregação dos do conjunto inicial.

Listing 2.6: Algoritmo de agregação de caminhos proposto em [Mud+09]

```
1
2  Sejam:
3  G = (V, E) - Grafo da rede
4  d - Nó de destino
5  k - Número de caminhos desejados entre cada nó da rede e d
6
7  Algoritmo PerDestinationSubGraphComputation(G,d,k)
8  SG = φ;
9
10 for v ∈ V do
11     P = P ∪ ComputePaths(G,v,d,k);
12 end
13
14 V_color = {v_1, v_2, ..., v_|P|};
15 E_color = φ;
16
17 for p_i, p_j ∈ P do
18     if SubGraphCompatible(p_i, p_j) == false
19         E_color = E_color ∪ {(v_i, v_j)};
20     end
21 end
22
23 (k, f) = VertexColoring((V_color, E_color));
24
25 SG = φ;
26 for 1 ≤ i ≤ k do
27     G_i = MergePaths({p_j ∈ P : f(v_j) = i});
28     SG = SG ∪ {G_i};
29 end
30
31 return SG;
```

Listing 2.7: Algoritmo de agregação de árvores proposto em [Mud+09]

```
1
2  Seja:
3  C = {G_1, G_2, ..., G_n} - Conjunto de sub-grafos
4
5  Algoritmo SubGraphMerge(C)
6  C' = φ;
7
8  for G_i ∈ C do //a permutação de C é aleatória
9      C = C - {G_i};
10     for G_j ∈ C do //a permutação de C é aleatória
11         G_m = G_i ∪ G_j;
12         if IsThereALoop(G_m) == false
13             C = C - {G_j};
14             G_i = G_m;
15         end
16     end
17     C' = C' ∪ {G_i};
18 end
19
20 return C';
```

2.4.2 Soluções alternativas

O problema de fundo em *MPLS* não é o de agregar caminhos em árvores mas sim o de diminuir o número de identificadores de *LSPs* diferentes que cada nó tem de suportar, como é assinalado pelos autores de [SFM08]. Na verdade, o conceito de *LSP m-t-p* é completamente ignorado pelos nós da rede, estes apenas conhecem identificadores e (re)escrita de identificadores para suporte à comutação de pacotes. Assim, para o sufixo comum entre caminhos é possível atribuir o mesmo *LSP*, mantendo no entanto *LSPs* diferentes nos outros troços dos caminhos.

Partindo desta observação, os autores apresentam um algoritmo [SFM08] de agregação de *LSPs p-t-p* em *LSPs m-t-p* para *MPLS* usando como ponto de partida a aproximação acima sugerida. Esta aproximação é específica e só pode ser utilizada em redes que implementam o multi-caminho com base em circuitos virtuais implementados como no caso do *MPLS*. Adicionalmente, os autores provam que o problema que resolvem é polinomial e que o algoritmo (polinomial) que propõem o resolve.

3

Algoritmo de Seleção de Caminhos

O algoritmo de seleção de caminhos [HMM13b] calcula um conjunto de caminhos entre um par de nós de entrada e saída da rede. Neste contexto, a rede é definida por um grafo $G = (V, E)$ simples (um grafo sem lacetes nem arcos paralelos), não orientado (onde (v_1, v_2) e (v_2, v_1) denotam o mesmo arco), conexo (existe caminho entre dois quaisquer nós) e pesado, sendo o peso de cada arco positivo.

3.1 Definições importantes

Seja $N \subseteq V$ o conjunto dos nós de entrada e saída da rede, ou seja, o conjunto dos nós que podem ser a origem ou o destino do tráfego. O custo de um caminho p (custo(p)) é a soma dos pesos dos arcos de p e o comprimento de p (compr(p)) é o número de arcos de p. Neste trabalho, todos os caminhos são simples, ou seja, todos os nós de um caminho são diferentes. Um conjunto de caminhos disjuntos (em termos de arcos) é um conjunto de caminhos cujos arcos são todos distintos. Dado um conjunto D de caminhos, a disjunção de D (disj(D)) é a maior cardinalidade dos subconjuntos de D que são conjuntos de caminhos disjuntos. Por exemplo, a disjunção de $A = \{1\,2\,4,\ 1\,2\,3\,5\,4,\ 1\,2\,6\,4,\ 1\,3\,2\,4,\ 1\,5\,4\}$ é três porque $\{1\,2\,6\,4,\ 1\,3\,2\,4,\ 1\,5\,4\}$ é um conjunto de caminhos disjuntos e há três caminhos em A que partilham o arco $(1, 2)$. Logo, só um deles pode pertencer a um conjunto de caminhos disjuntos.

Sejam $(x, y) \in N^2$ (com $x \neq y$) o par de nós para o qual se pretendem $k > 0$ caminhos distintos, $\mathcal{P}_{xy}$ o conjunto dos caminhos simples de x para y, que não é vazio porque o grafo é conexo, e $\mathcal{C}_{xy}$ o conjunto dos caminhos de custo mínimo (ou de menor custo) de x para y:

$$\mathcal{C}_{xy} = \{p \in \mathcal{P}_{xy} \mid (\forall p' \in \mathcal{P}_{xy})\ \text{custo}(p) \leq \text{custo}(p')\}.$$

selecCaminhos(k, h, f) =

$$\begin{cases} \text{melhorSubconj}(\mathcal{C}, k, \emptyset), & \text{se } |\mathcal{C}| \geq k; \\ \mathcal{I}_{h,f}, & \text{se } |\mathcal{C}| < k \text{ e } |\mathcal{I}_{h,f}| \leq k; \\ \mathcal{C} \cup \text{melhorSubconj}(\mathcal{I}_{h,f} \setminus \mathcal{C}, k - |\mathcal{C}|, \mathcal{C}), & \text{nos restantes casos.} \end{cases}$$

melhorSubconj(D, n, F) é um conjunto R que satisfaz as três propriedades:

- $R \in (D)_n$ (R é um subconjunto de D de cardinalidade n);
- $(\forall X \in (D)_n)$ disj$(R \cup F) \geq$ disj$(X \cup F)$;
- $(\forall X \in (D)_n)$ disj$(X \cup F) =$ disj$(R \cup F) \Rightarrow$ partilha$(R \cup F) \leq$ partilha$(X \cup F)$.

Figura 3.1: Conjunto dos caminhos selecionados entre um par de nós (x, y). $\mathcal{C}$ é o conjunto dos caminhos de menor custo e $\mathcal{I}_{h,f}$ é o conjunto dos caminhos interessantes de x para y.

Para simplificar a notação, omitem-se os nós origem e destino dos caminhos, escrevendo apenas $\mathcal{P}$ e $\mathcal{C}$

3.2 Descrição do algoritmo

O algoritmo que seleciona o melhor conjunto de caminhos entre os nós x e y (apresentado na Figura 3.2) pode ser dividido em três casos distintos:

- quando a cardinalidade de $\mathcal{C}$ é maior ou igual a k;
- quando a cardinalidade de $\mathcal{C}$ é menor que k e a cardinalidade de $\mathcal{I}_{h,f}$ (conjunto de caminhos interessantes de x para y) é menor ou igual a k;
- restantes casos.

No primeiro caso há pelo menos k caminhos de menor custo, retornando-se um "melhor subconjunto" de $\mathcal{C}$ de cardinalidade k. Melhor, neste caso, significa que maximiza a disjunção e que, de entre todos os subconjuntos cuja disjunção é máxima, minimiza a "partilha" de arcos entre os seus caminhos.

Para formalizar o conceito de partilha, sejam D um conjunto de caminhos e e um arco. A função penalização(D, e) atribui uma penalização ao uso (repetido) de e em D, onde ocorrências(D, e) denota o número de caminhos de D em que e ocorre:

$$\text{penalização}(D, e) = \begin{cases} 0, & \text{se ocorrências}(D, e) \leq 1; \\ (|D| + 1)^{\text{ocorrências}(D,e)}, & \text{se ocorrências}(D, e) \geq 2. \end{cases}$$

A partilha de arcos em D é a soma das penalizações atribuídas ao uso dos arcos do grafo em D:

$$\text{partilha}(D) = \sum_{e \in E} \text{penalização}(D, e).$$

Se $(D)_n$ representar o conjunto de todos os subconjuntos de D de cardinalidade n, o conjunto S retornado pelo algoritmo no caso que estamos a analisar verifica as três seguintes propriedades:

- $S \in (\mathcal{C})_k$;
- $(\forall X \in (\mathcal{C})_k)$ disj$(S) \geq$ disj(X);
- $(\forall X \in (\mathcal{C})_k)$ disj$(X) =$ disj$(S) \Rightarrow$ partilha$(S) \leq$ partilha(X).

Para exemplificar, considerem-se os dois subconjuntos de A de cardinalidade 4 cuja disjunção é 3: $B_1 = \{1\,2\,4,\ 1\,2\,6\,4\,1\,3\,2\,4,\ 1\,5\,4\}$ e $B_2 = \{1\,2\,3\,5\,4,\ 1\,2\,6\,4,\ 1\,3\,2\,4,\ 1\,5\,4\}$. Como partilha$(B_1) = 2 \times 5^2$ (porque os dois arcos de $1\,2\,4$ ocorrem duas vezes em B_1) e partilha$(B_2) = 3 \times 5^2$ (porque $(1, 2)$, $(2, 3)$ e $(5, 4)$ ocorrem duas vezes em B_2), o melhor subconjunto de A de cardinalidade 4 é B_1.

No segundo e no terceiro caso, é necessário considerar caminhos com custo superior ao mínimo, recorrendo-se à noção de caminho ótimo. Um caminho de x para y diz-se ótimo se tiver custo mínimo e comprimento igual ao menor comprimento dos caminhos de custo mínimo de x para y. Ou seja, o é um caminho ótimo se:

$$o \in \mathcal{C} \ \text{ e } \ (\forall p \in \mathcal{C})\ \text{compr}(o) \leq \text{compr}(p).$$

Os caminhos que podem ser selecionados designam-se por por caminhos interessantes. Estes são caminhos de menor custo, pertencentes a C ou caminhos próximos dos ótimos. Esta proximidade é caracterizada através de dois parâmetros, $h \geq 0$ e $f \geq 1$, da seguinte forma: a diferença entre o comprimento de um caminho próximo dos ótimos e o comprimento dos caminhos ótimos não excede h; e o quociente entre o custo de um caminho próximo dos ótimos e o custo dos caminhos ótimos não excede f. Seja o um caminho ótimo de x para y. O conjunto $\mathcal{I}_{h,f}$ dos caminhos interessantes de x para y é definido por:

$$\mathcal{I}_{h,f} = \mathcal{C} \cup \{p \in \mathcal{P} \mid \text{compr}(p) \leq \text{compr}(o) + h \ \wedge \ \text{custo}(p) \leq f \times \text{custo}(o)\}.$$

No segundo caso, o número total de caminhos interessantes não excede k. Neste caso o conjunto retornado é $I_{h,f}$. O conjunto selecionado terá, portanto, k ou menos caminhos (no caso deste número ser menor que k, o gestor da rede deverá ser informado para poder aumentar o valor h e f).

No terceiro caso, existem mais caminhos do que os necessários para satisfazer os requisitos. O conjunto S retornado tem então todos os caminhos de menor custo mais alguns dos interessantes restantes. Ou seja, é uma reunião da forma $S = \mathcal{C} \cup R$, onde R é um "melhor subconjunto" de $\mathcal{I}_{h,f} \setminus \mathcal{C}$ de cardinalidade $k - |\mathcal{C}|$. Ser melhor, neste caso, é maximizar a disjunção de S e, de entre os subconjuntos de $\mathcal{I}_{h,f} \setminus \mathcal{C}$ que a maximizam, minimizar a partilha de arcos em S.

É fácil verificar que as duas definições de "melhor subconjunto", usadas no primeiro e no último caso do algoritmo, podem ser unificadas. A função melhorSubconj(D, n, F), apresentada também na Figura 3.2, desempenha esse papel, onde D e F são conjuntos de caminhos e n é um inteiro positivo que não excede $|D|$.

3.3 Implementação do algoritmo

Na implementação do algoritmo, há três operações que merecem destaque: o cálculo dos caminhos de menor custo, o cálculo dos caminhos próximos dos ótimos e o cálculo de um melhor subconjunto. Durante esta secção, g é o maior grau de um nó do grafo.

3.3.1 Cálculo dos caminhos de menor custo

O conjunto dos caminhos de custo mínimo é obtido com uma variante do algoritmo de Dijkstra [Dij59] que guarda, para cada nó v, todos os nós que antecedem v nos caminhos de menor custo (até ao momento) de x para v. Por isso, o tempo gasto na (clássica) primeira fase é $O(|E| + |V| \log |V|)$, usando uma fila de Fibonacci [Cor+09], enquanto a geração dos caminhos tem complexidade temporal $O(|\mathcal{C}|\,|V|)$.

3.3.2 Cálculo dos caminhos próximos dos ótimos

Os caminhos próximos dos ótimos são gerados por força-bruta, iterando os sucessores diretos dos nós. Utiliza-se um vetor de booleanos para garantir que os caminhos são simples e pára-se assim que se pode concluir que os caminhos que estão a ser construídos teriam comprimento ou custo superior ao permitido. No pior caso portanto, esta operação é $O(g^{\min(\text{compr}(o)+h,\,|V|-1)})$, onde o é um caminho ótimo.

3.3.3 Cálculo de um melhor subconjunto

Na função melhorSubconj(D, n, F) geram-se subconjuntos de D de cardinalidade n para encontrar aquele que é retornado, tentando minimizar as computações relacionadas com o cálculo das disjunções. Para cada conjunto X gerado, começa-se por determinar, simultaneamente, o valor de partilha$(X \cup F)$ e o maior número z de caminhos de $X \cup F$ onde um mesmo arco ocorre ($z = \max_{e \in E}$ ocorrências$(X \cup F, e)$). Se $z = 1$, $X \cup F$ é um conjunto de caminhos disjuntos (logo, com a menor partilha possível) e a função melhorSubconj retorna X. Quando $z \geq 2$, sabe-se que disj$(X \cup F) \leq |X \cup F| - z + 1$, porque há $z - 1$ caminhos que não podem pertencer a um conjunto de caminhos disjuntos. Esta propriedade permite resolver com eficiência mais dois casos. O primeiro é quando $z = |X \cup F|$, que implica disj$(X \cup F) = 1$. O segundo faz uso do valor da disjunção obtido com o "melhor subconjunto" encontrado até ao momento. Denotando esse valor por $d_{\max}$, se $d_{\max} > |X \cup F| - z + 1$, não vale a pena calcular a disjunção de $X \cup F$, passando-se à geração e análise do próximo subconjunto de D. As ocorrências dos arcos do grafo (ocorrências$(X \cup F, e)$, para cada $e \in E$) são registadas numa matriz (triangular superior)

de inteiros, de $|V|$ por $|V|$. Consequentemente, o número de passos para inicializar e preencher a matriz e para calcular os valores de z e da partilha é de ordem $O(k\,|V| + |V|^2)$, usando o facto de $|X \cup F| = k$ em ambas as chamadas a melhorSubconj.

Quando é necessário saber se disj$(X \cup F) \geq d_{\max}$, geram-se subconjuntos de $X \cup F$ cujas cardinalidades variam de forma decrescente entre $k-z+1$ e, no pior caso, $d_{\max}$. Para cada conjunto Y gerado, verifica-se se todos os arcos dos caminhos de Y são distintos (recorrendo a uma matriz de booleanos de $|V|$ por $|V|$) e, em caso afirmativo, conclui-se imediatamente que disj$(X \cup F) = |Y|$. Se nenhum conjunto gerado for um conjunto de caminhos disjuntos, o valor de disj$(X \cup F)$ não chega a ser calculado, visto que é inferior a $d_{\max}$. Embora se restrinja o intervalo das cardinalidades dos conjuntos gerados, quando $z = 2$ e $d_{\max} = 1$, podem ser gerados $2^k - k - 1$ conjuntos (porque só não são gerados conjuntos de cardinalidades k e 0, e só é gerado um conjunto de cardinalidade 1). Portanto, no pior caso a complexidade temporal desta operação é $O(2^k\,(k\,|V| + |V|^2))$.

Somando os diferentes passos da função melhorSubconj, obtém-se uma expressão de ordem $O(\text{Comb}(|D|, n)\;2^k\,(k\,|V| + |V|^2))$, onde Comb$(i, j)$ são as combinações de i, j a j.

Em relação à complexidade temporal do algoritmo de seleção de caminhos, considera-se mais adequado defini-la em função dos conjuntos $\mathcal{C}$ e $\mathcal{I}_{h,f}$, e não apenas em função dos parâmetros de entrada. De facto, aqueles conjuntos só dependem do grafo, dos nós x e y e dos parâmetros h e f, e a sua utilização permite que as fórmulas finais não fiquem descaracterizadas (em consequência das inevitáveis maximizações). Analisando separadamente os três casos, denotando por α o comprimento máximo admitido para um caminho próximo dos ótimos ($\alpha = \min(h + \min_{p \in \mathcal{C}} \text{compr}(p),\ |V| - 1)$) e assumindo que $k \leq |V|$, conclui-se que, no pior caso, o número de passos da função selecCaminhos é de ordem:

$$\begin{cases} O(\text{Comb}(|\mathcal{C}|, k)\;2^k\;|V|^2), & \text{se } |\mathcal{C}| \geq k; \\ O(|E| + |V| \log |V| + |\mathcal{C}|\,|V| + g^\alpha), & \text{se } |\mathcal{C}| < k \text{ e } |\mathcal{I}_{h,f}| \leq k; \\ O(g^\alpha + \text{Comb}(|\mathcal{I}_{h,f}|, k)\;2^k\;|V|^2), & \text{nos restantes casos.} \end{cases}$$

Convém referir que, embora o algoritmo não seja polinomial, calcula os resultados em tempo útil nos contextos para os quais foi criado. O número de caminhos entre cada par de nós é de poucas unidades e o número de caminhos interessantes é controlado pelos outros dois parâmetros.

4

Redes de Computadores Utilizadas nos Testes Experimentais

4.1 Tipos e características das redes de computadores

Para avaliar os algoritmos de seleção e de agregação, os mesmos vão ser aplicados a diversas redes. A diversidade das redes selecionadas, bem como as características específicas de cada uma têm uma grande importância nos resultados obtidos e na análise destes e dos algoritmos. As redes podem ser dividas em dois grupos principais: regulares e não regulares. Adicionalmente, estas podem ser classificadas como sintéticas ou reais. As redes sintéticas são redes que não são baseadas na topologia de uma rede real. As redes reais são redes baseadas em redes que existem na realidade.

4.1.1 Redes regulares

As redes regulares são redes em que as configurações têm propriedades bem conhecidas. Estas são utilizadas, por exemplo, em ambientes de computação de alto desempenho, como centros de dados. Os caminhos interessantes nestas redes podem ser previamente definidos e descritos, tendo a aplicação do algoritmo de seleção a função de confirmar que foram selecionados os caminhos esperados. Serão utilizadas quatro redes regulares, que serão apresentadas mais adiante: *FullMesh*, Anel, *Fat Tree* e *Folded Clos*. Em todas será usado 1 como peso de cada canal. Devido a esta uniformidade de pesos, o parâmetro f perde relevância. No entanto, de forma a que o seu valor não limite os caminhos

selecionados, atribuiu-se a f o valor $h + 1$.[1]

4.1.2 Redes não regulares

As redes não regulares são redes que não exibem propriedades de regularidade nas relações entre os nós. Estas geralmente são redes reais, cuja topologia é influenciada pela geografia real dos componentes da rede. Uma grande vantagem da utilização deste tipo de redes é a aproximação à realidade, porque ao utilizá-las estão-se a testar os algoritmos em situações e ambientes reais. No entanto, a obtenção destas redes pode revelar-se complicada. Estas são geralmente redes de operadores privados que não têm interesse em revelar a sua topologia real, por razões empresariais. Existem também redes não regulares sintéticas, geralmente geradas por algoritmos, como são os casos das redes aleatórias. Sob alguns pontos de vista, as redes aleatórias também poderiam ser consideradas regulares. Por exemplo, o grau dos nós pode seguir uma dada distribuição.

Foram utilizadas quatro redes não regulares, três *backbones* reais e uma rede aleatória. Em todas foram removidos canais paralelos e nós que não acrescentavam diversidade. Nas redes reais o peso dos canais usado é proporcional à distância entre os nós e na rede aleatória o peso dos canais é 1, tal como nas redes regulares.

4.1.3 Características das redes

As características específicas das redes trazem diversidade ao conjunto das redes selecionadas e influenciam os resultados empíricos das execuções dos algoritmos. Nas redes regulares algumas características são mais evidentes, pois muitas vezes a própria rede foi desenhada para ter essas características. Nas não regulares é mais complicado determinar características ou padrões específicos, porque estas dependem de fatores de natureza geográfica, económica, política ou simplesmente aleatória, que influenciam o seu planeamento e construção. Ainda assim, é importante compreender as características específicas de cada rede, de forma a analisar em contexto os resultados obtidos. Serão apresentadas de seguida algumas características interessantes das redes.

- Grau de um nó — O grau de um nó de uma rede indica qual o número de canais a que este está ligado. O grau médio dos nós da rede influencia a redundância de caminhos possível nesta. Quanto maior o grau médio, maior será a redundância possível.

- Variação do custo dos canais — Este valor representa o intervalo dentro do qual se encontram os custos dos canais da rede. No caso de existirem canais com custo muito baixo e outros com custo muito elevado numa rede, a escolha do parâmetro f (variação do custo) pode tornar-se complicada, podendo prejudicar de forma crítica as possibilidades de escolha do algoritmo para certos pares.

[1] Quando o peso dos arcos é 1, o custo de um caminho é igual ao seu comprimento. Portanto, $\text{custo}(p) \leq \text{custo}(o) + h$ é equivalente a $\text{custo}(p)/\text{custo}(o) \leq 1 + h/\text{custo}(o) \leq 1 + h$.

- Número de caminhos de menor custo entre um par de nós — Representa o número total de caminhos de menor custo existentes entre um par de nós. O seu peso no conjunto total dos caminhos selecionados para esse par pode influenciar a qualidade do conjunto.
- Número de caminhos disjuntos entre um par de nós — Representa o número máximo de caminhos disjuntos dentro de um conjunto de caminhos entre um par de nós. Este valor define a tolerância a falhas garantida por esse conjunto.
- Número de caminhos interessantes entre um par de nós — Representa o universo de caminhos que são interessantes para serem selecionados entre um par de nós, ou seja, cuja qualidade em função de diferentes parâmetros torna o seu uso interessante. Por exemplo, certos caminhos, por serem demasiado longos ou de custo demasiado elevado, não têm qualidade suficiente para serem utilizados, não sendo portanto interessantes.

As características das redes utilizadas nos testes (que serão de seguida detalhadas) são apresentadas na tabela 4.1.

4.2 Redes regulares escolhidas

4.2.1 Full Mesh

Uma rede *Full Mesh* de n nós é uma rede em que cada nó tem grau $n-1$, estando diretamente ligado a todos os outros por um canal, formando uma *clique*. Entre cada par de nós apenas existe um caminho mais curto, de comprimento 1. Adicionalmente, existe uma elevada redundância em termos de caminhos na rede, tendo cada par de nós $n-1$ caminhos disjuntos entre si em termos de arcos e vértices. Estes caminhos têm comprimento 1 ou 2 e, com exceção do de menor custo, utilizam todos um nó intermédio para atingir o destino, sendo muito adequados para uma distribuição equitativa de carga. Outros caminhos já não têm interesse. Assim, para cobrir todos os caminhos interessantes entre cada par de nós, 1 caminho de menor custo e mais $n-2$ disjuntos, é necessário considerar um $k = n-1$. Adicionalmente, sendo que no máximo os caminhos têm 1 nó intermédio, basta utilizar um h de valor 1. A *Full Mesh* utilizada, representada na figura 4.1(a), tem 6 nós, 15 canais, e todos os nós são origem e destino de tráfego. A seleção de caminhos é realizada com valores de $k = 5$, $h = 1$ e $f = 2$.

4.2.2 Anel

Uma rede em anel com n nós, como o nome indica, tem uma topologia aproximada à de um anel, em que cada nó tem grau 2, estando ligado apenas a 2 nós diferentes. Nesta topologia apenas existem 2 caminhos entre cada dois nós. No máximo, um caminho pode ter $n-1$ de comprimento, o que obriga a uma extensão de comprimento permitida em

relação ao mais curto de $n-2$. Assim, quanto maior for o número de nós da rede, maior terá de ser a extensão de comprimento permitida de forma a cobrir todos os caminhos. Foi utilizada a rede representada na figura 4.1(b), com 10 nós, 10 canais, e em que todos os nós são origem e destino de tráfego. A seleção de caminhos é realizada com valores de k = 2, $h = 8$ e $f = 9$.

4.2.3 Fat Tree

Uma rede *Fat Tree* é uma rede hierárquica frequentemente usada em centros de dados e é definida por 3 camadas distintas, que formam uma rede com um aspeto semelhante ao de uma árvore. No fundo da árvore estão os *Top of Rack switches* (*ToRs*) que fazem a ligação entre os servidores de cada *rack* e a rede. Na 1ª camada (imediatamente acima da anterior) estão os *aggregation switches* que oferecem redundância ligando os *ToRs* às camadas superiores. Cada *ToR* para garantir a tolerância a 1 falha liga-se a 2 *aggregation switches*. Na 2ª camada estão os *access routers* que permitem redundância e suporte para o volume de tráfego pedido pelas camadas inferiores. Estes fazem a ligação entre os *aggregation switches* e a 3ª camada. Nesta última camada estão os *core routers* que fazem a ligação com os *access routers*. É nesta última camada que é gerida a entrada e saída de tráfego dos centros de dados.

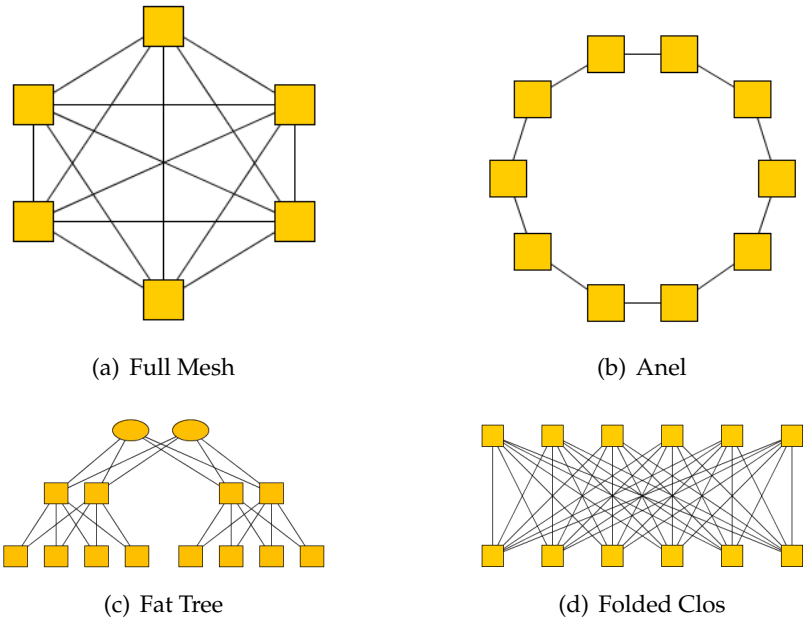
Foi utilizada uma rede simplificada, representada na figura 4.1(c). Nesta rede colapsámos a parte que se liga ao exterior e suprimimos os *ToR swicthes* que não encaminham tráfego entre si. Consideramos que apenas os *aggregation switches* originam ou terminam tráfego. Esta rede tem 14 nós e 24 canais. Os nós são 8 *aggregation switches*, 4 *access routers* e 2 *core routers*. O número de caminhos de menor custo entre cada par de nós (*aggregation switches*) pode ser dado pela expressão 2^{2m-1}, sendo m o número de níveis que é necessário subir para chegar ao nó de destino. Por exemplo, entre 2 *aggregation switches* filhos do mesmo *access router*, o tráfego apenas necessita de subir 1 nível, sendo então o número de caminhos de menor custo entre esses dois nós $2^{2\times1-1}$ = 2. Desta forma, para cobrir todos os caminhos de menor custo entre *aggregation switches* da rede (os únicos interessantes), k pode tomar valor 2^{2m-1}, sendo m neste caso o número de níveis na rede. Adicionalmente, como apenas serão selecionados caminhos de menor custo, o parâmetro h pode tomar valor 0. A seleção de caminhos é realizada com valores de k = 2 ou k = 8, $h = 0$ e $f = 1$.

4.2.4 Folded Clos

Esta topologia é frequentemente proposta para redes de centros de dados como alternativa à *Fat Tree* tradicional, introduzindo mais redundância no encaminhamento. É definida por duas camadas distintas, formando um grafo bipartido em que cada nó da primeira camada está ligado a todos da segunda. Tal como na *Fat Tree*, no fundo do grafo encontram-se os *ToRs*. Estes ligam à primeira camada, onde se encontram os *aggregation switches*. Cada *ToR* liga-se a dois *aggregation switches*. Na segunda camada encontram-se

os *intermediate switches* aos quais os *aggregation switches* se ligam. Os *intermediate switches* ligam-se à rede, gerindo a entrada e saída de tráfego do centro de dados.

Foi utilizada uma rede simplificada, representada na figura 4.1(d), sem *ToR switches* e em que apenas os *aggregation switches* são origem e destino de tráfego. Esta rede tem 12 nós e 36 canais. Os nós são 6 *aggregation switches* e 6 *intermediate switches*, estando cada *aggregation switch* diretamente ligado aos 6 *intermediate switches*. Os caminhos interessantes nesta rede são simultaneamente todos de menor custo e disjuntos entre si. O seu número é igual ao número de *intermediate switches*. Assim, para os selecionar basta utilizar-se um k de valor igual ao número de *intermediate switches* e um h de valor 0, pois todos são de menor custo. A seleção de caminhos é realizada com valores de k = 6, $h = 0$ e $f = 1$.



(a) Full Mesh (b) Anel

(c) Fat Tree (d) Folded Clos

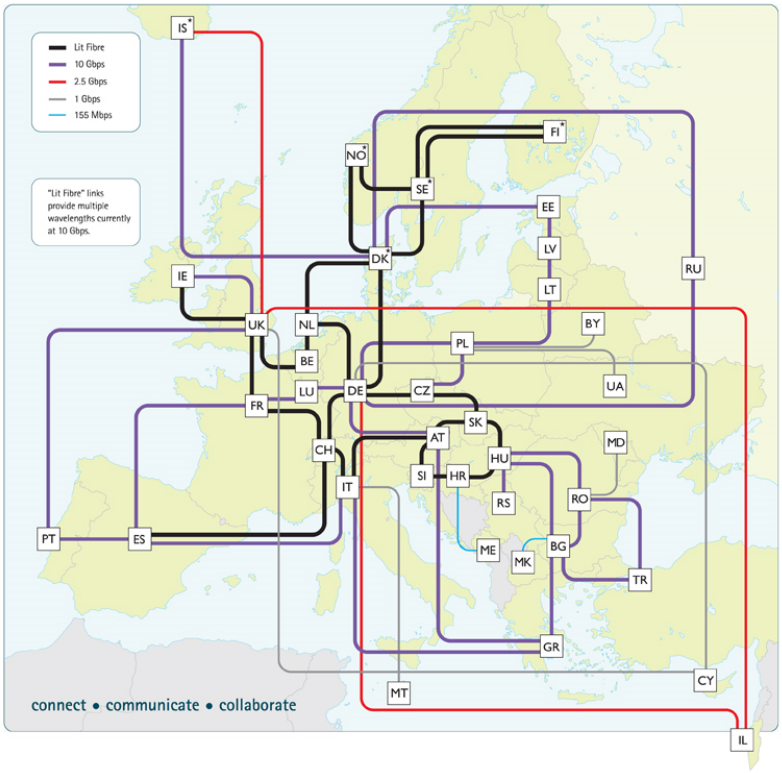
Figura 4.1: Redes regulares utilizadas

4.3 Redes não regulares escolhidas

4.3.1 GÉANT

A *GÉANT*, representada na figura 4.2, é uma rede europeia que interliga as diferentes redes de investigação e educação dos países europeus. A topologia da rede é subordinada a critérios geo-políticos, existindo um nó por cada país europeu que participa na rede. Adicionalmente, também a distribuição de canais na rede é afetada por fatores geo-políticos e económicos, havendo um maior grau médio nos nós do centro e norte da Europa em comparação com os restantes. A configuração da rede utilizada tem 31 nós e 50 canais.

Figura 4.2: Topologia original da rede GÉANT



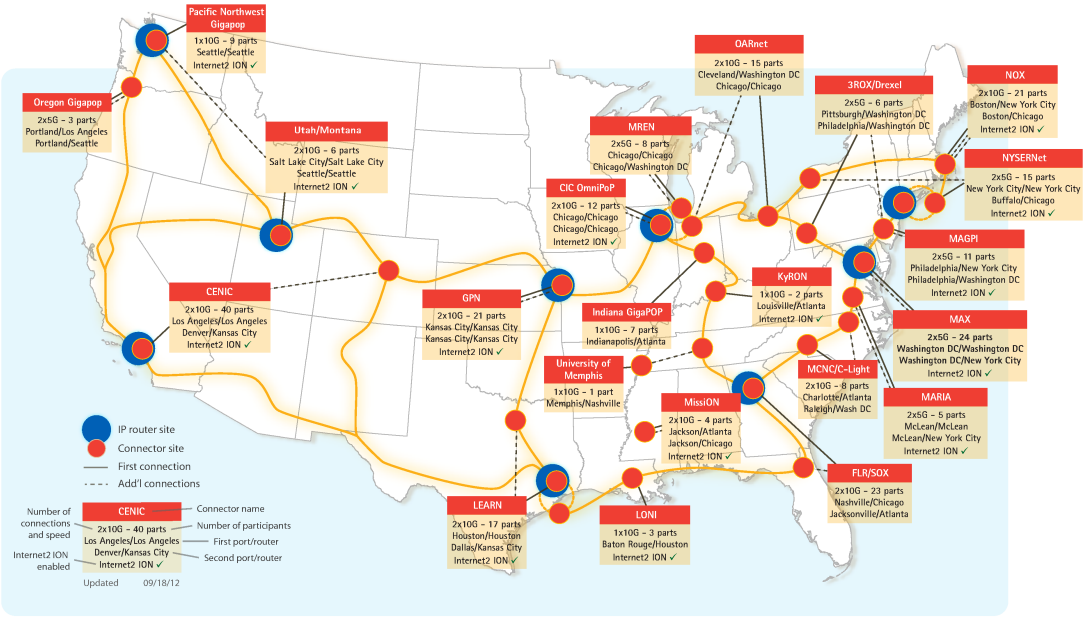
4.3.2 Internet2

A *Internet2*, representada na figura 4.3, é um consorcio norte-americano que liga universidades, centros de investigação, corporações e outras redes regionais de educação e investigação nos Estados Unidos da América. Tal como na *GÉANT*, a topologia da rede tem uma grande influência económica, existindo um nó nas cidades de maior importância. Foi realizada uma interpretação do mapa da rede disponibilizado ao público, dividindo os nós em 2 tipos distintos: nós que participam no tráfego que não os envolve diretamente e nós que apenas originam ou terminam tráfego, não fazendo trânsito. Os canais relacionados com os primeiros têm um valor que representa a latência, enquanto os canais relacionados com os segundos têm um valor muito mais elevado, de forma a evitar que seja feito trânsito através desses nós. A configuração da rede utilizada tem 25 nós e 44 canais.

4.3.3 NTT Communications

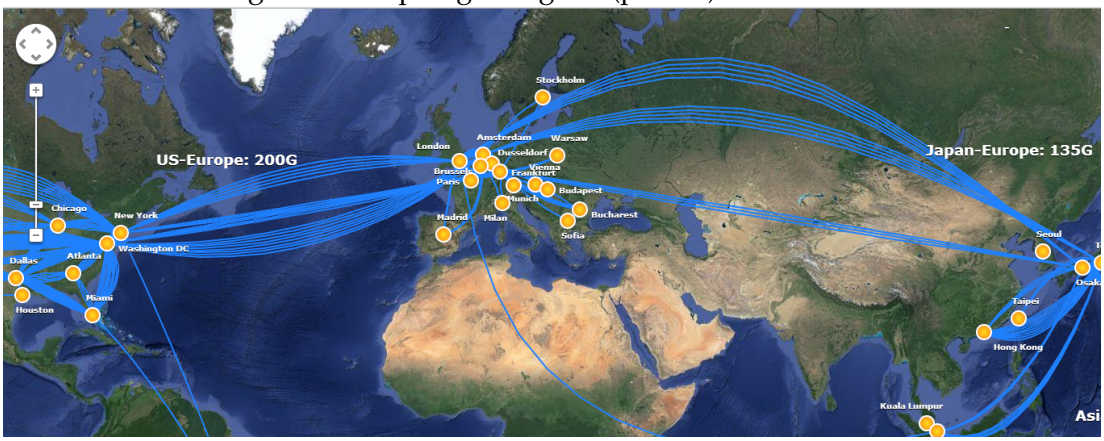
A *NTT Communications* é uma corporação subsidiária da *NTT* (*Nippon Telegraph and Telephone*) e possui uma rede IP, representada na figura 4.4, com uma distribuição geográfica mundial. Por se tratar de uma rede comercial, fatores económicos influenciam a topologia da mesma, existindo uma grande concentração de nós e canais nas zonas dos E.U.A., Europa e extremo oriente, por oposição às outras zonas do globo que possuem menos nós

Figura 4.3: Topologia original da rede Internet2



e um menor grau de ligações. A configuração da rede utilizada tem 27 nós e 63 canais.

Figura 4.4: Topologia original (parcial) da rede NTT



4.3.4 Aleatória

Uma rede aleatória é gerada através de um processo pseudo-aleatório geralmente com base numa certa distribuição estatística. Para gerar uma rede destas apenas foi indicado o número de nós desejado e a distribuição do grau. Para os testes empíricos foi gerada uma rede aleatória com base numa distribuição de *Poisson*, com 30 nós, cujo grau médio é 3. A rede foi gerada utilizando uma biblioteca publicamente disponibilizada, acessível através do site do projeto GraphStream (`http://graphstream-project.org/doc/Generators/Random-graph-generator_1.0`).

Tabela 4.1: Conjunto das redes utilizadas e as suas características.

Redes	# de nós	# de canais	grau médio	custo dos canais	# de caminhos de menor custo
Full Mesh	6	15	5	1	15
Anel	10	10	2	1	50
Fat Tree	14	24	3.4	1	152
Folded Clos	12	36	6	1	90
GÉANT	31	50	3.2	[2.3, 71.1]	817
Internet 2	25	44	3.5	[6.6, 44.1]	436
NTT	27	63	4.6	[8.7, 603.7]	664
Aleatória	30	48	3.2	1	748

5

Análise Empírica do Algoritmo de Seleção de Caminhos

Com o objetivo de analisar a qualidade do algoritmo de seleção e dos caminhos por ele selecionados, este foi executado para as redes anteriormente descritas. Adicionalmente, o algoritmo SPAIN [Mud+09; Mud+10] também foi executado para as mesmas redes com o mesmo parâmetro k, de forma a permitir uma comparação entre este e o nosso. Todos os testes foram executados num computador com 4 Gbytes de RAM, CPU a 2,53 GHertz e usando a Java VM versão 1.7. Para cada rede, o algoritmo foi executado para todos os pares $(x, y) \in N^2$ tais que $x < y$.

5.1 Parametrização

Tal como explicado no capítulo 3, o nosso algoritmo de seleção recebe 6 parâmetros distintos: a rede sobre a qual vai executar, os nós x e y para os quais serão calculados os caminhos, o número de caminhos desejados entre o par de nós (k), a extensão máxima do comprimento de um caminho em relação ao comprimento mínimo dos de menor custo do par de nós (h) e a extensão máxima do custo de um caminho em relação ao custo dos caminhos de menor custo do par de nós (f).

Para as redes regulares, os parâmetros utilizados em cada uma já foram apresentados no capítulo anterior. Estes são os necessários para englobar todos os caminhos considerados interessantes. No entanto, nas redes não regulares, devido à difícil caracterização de propriedades para cada uma, torna-se complicado inferir *a priori* quais os caminhos interessantes. Assim, foi utilizado um conjunto de parâmetros que servem para demonstrar certas características do algoritmo, em contexto com cada uma das redes, e que permitem

Tabela 5.1: Resultados da execução do algoritmo nas redes regulares.

	Algoritmo		SPAIN	
Redes	# mínimo de falhas cobertas	# máximo de caminhos de menor custo	# mínimo de falhas cobertas	# máximo de caminhos de menor custo
FullMesh	4	1	4	1
Anel	1	2	1	2
Fat Tree	1	8	1	4
Folded Clos	5	6	5	6

uma análise comparativa em relação ao algoritmo SPAIN.

5.2 Análise dos resultados nas redes regulares

Os resultados das execuções do nosso algoritmo e do algoritmo SPAIN, para as redes regulares, são apresentados na tabela 5.1. Estes são compostos por dois dados: número mínimo de falhas cobertas e número máximo de caminhos de menor custo. O primeiro indica o número mínimo de falhas toleradas por todos os pares de nós da rede que são origem e destino de tráfego. Este valor corresponde ao número mínimo de caminhos disjuntos em termos de canais da rede entre cada par de nós menos 1. Por exemplo, se entre um par de nós existirem 3 caminhos disjuntos, isto significa que este conjunto de caminhos permite suportar 2 falhas simultâneas de canais. O segundo indica o número máximo de caminhos de menor custo selecionados entre um par de nós.

Os resultados permitem verificar que o nosso algoritmo selecionou, para cada rede regular, os caminhos esperados, ou seja, todos os previamente considerados interessantes. A execução do algoritmo para cada rede (isto é, para todos os pares $(x, y) \in N^2$ tais que $x < y$) não excedeu 140 milissegundos.

O algoritmo do SPAIN apresenta resultados semelhantes, exceto com a rede *fat tree* (figura 4.1(c)) quando os caminhos de x para y têm de subir dois níveis. Nesses casos, $k = 8$ (porque há 8 caminhos ótimos), no entanto, o conjunto dos caminhos selecionados tem cardinalidade quatro. Este comportamento deve-se ao critério de paragem do algoritmo. O SPAIN aplica o algoritmo de Dijkstra para descobrir um caminho de custo mínimo de x para y. Depois, aumenta significativamente os pesos dos arcos desse caminho (adicionando-lhes o número de arcos do grafo, quando todos os pesos iniciais são 1). Estes dois passos são repetidos até terem sido descobertos k caminhos de x para y ou até o caminho obtido ser igual a um dos previamente descobertos. Ora, nos casos que estamos a analisar, os quatro primeiros caminhos encontrados têm a forma: $x\, v_1\, a\, w_1\, y$, $x\, v_2\, a\, w_2\, y$, $x\, v_1\, b\, w_1\, y$ e $x\, v_2\, b\, w_2\, y$, onde v_1 e v_2 representam os nós adjacentes a x, w_1 e w_2 representam os nós adjacentes a y, e a e b denotam os nós do nível de cima. Nesse momento, os arcos incidentes em x ou y têm todos peso 49 $(1 + 24 + 24)$ e os incidentes em a

Tabela 5.2: Características das redes não regulares.

Redes	# de nós	# de canais
GÉANT	31	50
Internet 2	25	44
NTT	27	63
Aleatória	30	48

ou b têm peso 25. O problema é que o custo atual de qualquer um dos 8 caminhos ótimos é o mesmo e, como o algoritmo de Dijkstra é determinista, o quinto caminho obtido é igual ao primeiro descoberto. Os próprios autores reconhecem esta limitação ([Mud+09], secção 8.3).

5.3 Análise dos resultados nas redes não regulares

Os resultados das execuções do nosso algoritmo e do algoritmo SPAIN, para as redes não regulares (caracterizadas no capítulo anterior e na tabela 5.2), são apresentados em diferentes tabelas. Nestas redes, devido à dificuldade de definir de forma exata quais os caminhos interessantes, tal como explicado anteriormente, foram utilizados parâmetros que permitissem a obtenção de resultados bons para analisar e discutir as propriedades dos algoritmos. Os parâmetros utilizados no nosso algoritmo para as redes reais foram $k = 3$, $h = 5$ e $f = 5$. Na rede aleatória, devido ao facto de todos os arcos terem peso 1, f tomou o valor $h + 1$, ou seja, 6, tal como aconteceu nas redes regulares. No algoritmo SPAIN foi utilizado $k = 3$.

Os resultados apresentados nas tabelas 5.3 e 5.4 são os do nosso algoritmo, nas 5.5 e 5.6 os do algoritmo SPAIN. A primeiras tabela de cada algoritmo (tabelas 5.3 e 5.5) apresenta os dados e estatísticas diretamente relacionados com os caminhos selecionados: o número total de caminhos selecionados pelo algoritmo, a distribuição do número de caminhos selecionados em função dos pares de nós e a cobertura dos caminhos de menor custo. Os primeiros dados indicam o número total de caminhos selecionados pelo algoritmo no conjunto de todos os pares de nós de origem e destino de tráfego. Os segundos indicam a percentagem de pares para os quais foram selecionados 1, 2 e 3 caminhos. Por fim, os terceiros indicam a percentagem de caminhos de menor custo existentes na rede que foi selecionada pelo algoritmo. A segunda tabela de cada algoritmo (tabelas 5.4 e 5.6) apresenta dados sobre a tolerância a falhas, indicando a percentagem de pares que suportam um certo número de falhas.

Os resultados do nosso algoritmo mostram que este seleciona todos os caminhos de menor custo que existem entre qualquer par de nós, com exceção da rede aleatória. Nesta, devido ao facto de existirem pares com mais de 3 caminhos de menor custo, nem todos

Tabela 5.3: Resultados da execução do algoritmo nas redes não regulares (Caminhos selecionados) com $k = 3$, $h = 5$ e $f = 5$.

Redes	# Caminhos selecionados	% de pares em que foram selecionados			% de caminhos de menor custo selecionados
		1 caminho	2 caminhos	3 caminhos	
GÉANT	1375	1.29	1.72	96.99	100
Internet 2	897	0.33	0.33	99.33	100
NTT	1041	0.85	1.71	97.44	100
Aleatória	1304	0.0	0.23	99.77	90.64

Tabela 5.4: Resultados da execução do algoritmo nas redes não regulares (Tolerância de falhas) com $k = 3$, $h = 5$ e $f = 5$.

Redes	% de pares que suportam		
	0 falhas	1 falha	2 falhas
GÉANT	1.29	75.7	23.01
Internet 2	0.33	98.67	1.0
NTT	0.85	68.95	30.2
Aleatória	2.99	60.23	36.78

Tabela 5.5: Resultados da execução do SPAIN nas redes não regulares (Caminhos selecionados) com $k = 3$.

Redes	# Caminhos selecionados	% de pares em que foram selecionados			% de caminhos de menor custo selecionados
		1 caminho	2 caminhos	3 caminhos	
GÉANT	1336	0.0	12.69	87.31	95.71
Internet 2	806	0.0	31.33	68.67	100.0
NTT	1015	0.0	10.83	89.17	95.45
Aleatória	1284	0.0	4.83	95.17	75.13

Tabela 5.6: Resultados da execução do SPAIN nas redes não regulares (Tolerância de falhas) com $k = 3$.

Redes	% de pares que suportam		
	0 falhas	1 falha	2 falhas
GÉANT	0.0	74.19	25.81
Internet 2	0.0	88.0	12.0
NTT	0.0	65.81	34.19
Aleatória	0.0	51.72	48.28

são selecionados, pois o k utilizado tem valor 3. Assim, para se obterem todos os caminhos de menor custo na rede aleatória, é necessário atribuir a k um valor superior a 3 quando se selecionam os caminhos entre aqueles pares.

Ao nível da tolerância a falhas, os resultados do nosso algoritmo mostram que nem todos os pares de caminhos suportam pelo menos uma falha, existindo uma pequena fração destes que não suporta nenhuma. Esta, nas redes reais, coincide com a fração de pares para os quais apenas é selecionado 1 caminho. Estes são casos excecionais, bem identificados, que acontecem quando só há um caminho de menor custo que tem um comprimento ou um custo muito baixo e os caminhos alternativos, por oposição, têm um comprimento ou um custo muito elevado (não sendo portanto considerados próximos dos ótimos). Por exemplo, se o caminho de menor custo de um certo par tiver comprimento 1 e todas as alternativas tiverem comprimento maior que $h + 1$. Aumentando os parâmetros h ou f, consegue-se eliminar estas exceções e garantir que todos os pares toleram pelo menos 1 falha.

Comparativamente, o SPAIN privilegia a tolerância a falhas face à distribuição de carga e à qualidade dos caminhos. Isto é, seleciona um conjunto de caminhos que garante a tolerância a pelo menos uma falha, embora não selecione todos os melhores caminhos e não faça um controlo da qualidade dos selecionados. Por outro lado, e tal como já tinha acontecido nas redes regulares, o algoritmo seleciona menos caminhos (como se observa comparando nas tabelas a coluna do número de caminhos selecionados e as colunas da distribuição do número de selecionados). Tal deve-se à estratégia seguida pelo algoritmo e não tem solução. Ou seja, quando o algoritmo pára a geração de caminhos devido à utilização prévia dos arcos, alterar o valor do parâmetro k não resolve a situação.

No nosso algoritmo é possível solucionar os casos excecionais. Para tal executou-se o algoritmo para todos os pares de nós com um valor de f (o parâmetro que no caso destas redes provocava as exceções) que permitiria cobrir pelo menos um segundo caminho para os pares excecionais (Geant com $f = 12$, Internet2 com $f = 7$ e NTT com $f = 14$). Os resultados destas novas execuções são apresentados nas tabelas 5.7 e 5.8. Estes mostram que os casos excecionais desapareceram, para todos os pares são selecionados pelo menos 2 caminhos e todos toleram pelo menos 1 falha.

Tabela 5.7: Resultados da execução do algoritmo nas redes não regulares com parâmetros alargados (Caminhos selecionados).

Redes	# Caminhos selecionados	% de pares em que foram selecionados			% de caminhos de menor custo selecionados
		1 caminho	2 caminhos	3 caminhos	
GÉANT	1391	0.0	0.86	99.14	100
Internet 2	899	0.0	0.33	99.67	100
NTT	1053	0.0	0.0	100	100

Tabela 5.8: Resultados da execução do algoritmo nas redes não regulares com parâmetros alargados (Tolerância de falhas).

Redes	% de pares que suportam		
	0 falhas	1 falha	2 falhas
GÉANT	0.0	75.91	24.09
Internet 2	0.0	99.0	1.0
NTT	0.0	68.66	31.34

Esta solução, no entanto, traz algumas desvantagens, porque se está a eliminar uma barreira que fazia controlo de qualidade na escolha dos caminhos. Ou seja, ao se aumentar o valor de h ou de f, corre-se o risco de deixar o algoritmo escolher caminhos que anteriormente não seriam interessantes, para pares não excecionais (com tolerância a pelo menos 1 falha). Adicionalmente, ao aumentar o universo dos caminhos interessantes, está-se potencialmente a aumentar o tempo de execução do algoritmo para todos os pares, quando apenas seria necessário a aplicação dos novos parâmetros aos pares de exceção. Como alternativa, criou-se uma versão alterada do algoritmo de seleção. Nesta, após a seleção dos caminhos para um par, se o par não tolerar pelo menos 1 falha, é selecionado um caminho disjunto em relação a um caminho ótimo, com o menor custo possível. Assim, é possível resolver o problema de um k, h ou f limitado nos casos excecionais, sem prejudicar a seleção de caminhos nos outros pares. Os resultados desta versão do algoritmo são apresentados em 5.9 e 5.10. No caso da rede aleatória, esta versão do algoritmo selecionou 4 caminhos para os pares excecionais. Isto justifica o facto da soma das percentagens das colunas relativas ao número de caminhos selecionados para cada par não ser 100%, pertencendo a percentagem restante aos pares onde foram selecionados 4 caminhos.

Tabela 5.9: Resultados da execução do algoritmo alterado nas redes não regulares (Caminhos selecionados).

Redes	# Caminhos selecionados	% de pares em que foram selecionados			% de caminhos de menor custo selecionados
		1 caminho	2 caminhos	3 caminhos	
GÉANT	1381	0.0	3.01	96.99	100
Internet 2	898	0.0	0.67	99.33	100
NTT	1044	0.0	2.56	97.44	100
Aleatória	1318	0.0	0.23	96.55	90.64

Tabela 5.10: Resultados da execução do algoritmo alterado nas redes não regulares (Tolerância de falhas).

Redes	% de pares que suportam		
	0 falhas	1 falha	2 falhas
GÉANT	0.0	76.99	23.01
Internet 2	0.0	99.0	1.0
NTT	0.0	69.52	30.48
Aleatória	0.0	63.45	36.55

5.4 Tempos de execução

Ao nível dos tempos de execução o algoritmo do Spain calculou os resultados apresentados para cada uma das redes em menos de 350 milissegundos. O nosso algoritmo (versão original e versão com tratamento de exceções) obteve os resultados em menos de 120 segundos, exceto para a NTT. Neste caso a execução completa demorou cerca de 4 horas. Nesta rede, há aproximadamente 35 pares com uma, duas ou três dezenas de milhares de caminhos interessantes, para cada um dos quais o tempo médio de execução foi de cerca de 5 minutos (e o tempo máximo inferior a 20 minutos). Curiosamente, o maior conjunto de caminhos interessantes (com $29\,416$ elementos) foi processado em menos de um segundo.

5.5 Análise geral

Em resumo, o nosso algoritmo, com os parâmetros adequados, seleciona os caminhos interessantes, isto é, tenta satisfazer os dois objetivos: tolerância a falhas e distribuição de carga. A tolerância a falhas ao privilegiar a disjunção entre os caminhos e a distribuição de carga ao escolher de forma prioritária os caminhos de menor custo. Adicionalmente, o algoritmo permite facilmente classificar a qualidade dos caminhos selecionados para

cada par, porque comparara o valor de k com a cardinalidade do conjunto dos caminhos de custo mínimo e com a cardinalidade do conjunto dos caminhos interessantes, calculando depois a disjunção do conjunto retornado. A parametrização também permite uma escolha mais controlada dos caminhos em termos de qualidade, ao limitar a sua extensão e custo relativo.

Esta capacidade de análise do algoritmo permite a sua modificação e adaptação com facilidade, o que permitiu, por exemplo, a criação da versão deste com tratamento das exceções.

6

Algoritmo de Agregação de Caminhos em Árvores

O algoritmo de agregação de caminhos criado [HMM13a] agrega um conjunto de caminhos num conjunto de árvores. Como o objetivo da agregação de caminhos é diminuir o custo espacial da parametrização do encaminhamento nos nós da rede, o algoritmo tem como orientação principal a agregação dos caminhos no menor número de árvores possível. Este algoritmo é determinista. Partindo de um conjunto vazio de árvores, vão-se agregando caminhos às árvores existentes, criando uma nova árvore apenas quando a inserção dos caminhos em qualquer uma das árvores que já existem resultaria num grafo cíclico ou não conexo. A grande diferença em relação aos algoritmos semelhantes descritos no capítulo do trabalho relacionado 2.4 é que se começa por inserir "pares de caminhos compatíveis", por uma ordem e numa árvore que tentam ser adequadas às futuras agregações.

6.1 Definições importantes

Por abuso, chamamos *par* a um conjunto $\{p, q\}$ com dois caminhos, denotando-o por (p, q), mas os caminhos são sempre distintos e a ordem pela qual ocorrem é irrelevante. S é um conjunto de caminhos num grafo G. $G_p = (V_p, E_p)$ é um grafo composto por um caminho $p \in S$ e uma árvore t de G é um sub-grafo de G conexo e acíclico (definido por (V_t, E_t)).

A noção de "compatibilidade" é fundamental e define-se entre dois caminhos, entre um caminho e uma árvore e entre um par de caminhos e uma árvore. No primeiro caso, a compatibilidade será usada para definir a ordem pela qual os pares de caminhos são

processados. Nos dois últimos, será usada para escolher a árvore onde um caminho ou um par de caminhos é inserido, quando há várias alternativas.

A $compatibilidade$ entre um par de caminhos (p, q) é -1 se o grafo $(V_p \cup V_q, E_p \cup E_q)$ for cíclico; e é $|V_p \cap V_q|$, o número de nós em comum, no caso contrário. A $compatibilidade$ entre um caminho p e uma árvore t define-se de forma semelhante: é -1 se o grafo $(V_t \cup V_p, E_t \cup E_p)$ for cíclico; e é $|V_t \cap V_p|$, no caso contrário. Por último, a $compatibilidade$ entre um par de caminhos (p, q) e uma árvore t é -1 se o grafo $(V_t \cup V_p \cup V_q, E_t \cup E_p \cup E_q)$ for cíclico; e é $|V_t \cap V_p| + |V_t \cap V_q|$, nos outros casos. Dois caminhos (respetivamente, um caminho e uma árvore, ou um par de caminhos e uma árvore) dizem-se *compatíveis* se a compatibilidade entre eles — denotada por compat$(\cdot, \cdot)$ — for positiva.

Note-se que dois caminhos sem nós em comum (respetivamente, um caminho sem nós em comum com uma árvore) não são compatíveis, porque a reunião dos respetivos grafos não é um grafo conexo. Mas um par de caminhos pode ser compatível com uma árvore, mesmo que um dos caminhos não partilhe qualquer nó com esta (desde que o outro partilhe). As seguintes propriedades, que permitem efetuar operações com entidades compatíveis, são fáceis de verificar:

- Se (p, q) for um par de caminhos compatíveis, a criação de um novo grafo com (p, q), definido por $(V_p \cup V_q, E_p \cup E_q)$, produz uma árvore;
- Se o caminho p e a árvore t forem compatíveis, a inserção de p em t, que transforma t no grafo $(V_t \cup V_p, E_t \cup E_p)$, produz uma árvore;
- Se (p, q) for um par de caminhos compatíveis, t for uma árvore, e (p, q) e t forem compatíveis, a *inserção* de (p, q) em t, que transforma t no grafo $(V_t \cup V_p \cup V_q, E_t \cup E_p \cup E_q)$, produz uma árvore.

6.2 Descrição do algoritmo

O algoritmo de agregação de caminhos (esquematizado na listagem 6.1) é composto por três fases: geração dos pares de caminhos compatíveis e dos caminhos singulares iniciais (os caminhos de S que são incompatíveis com todos os outros e que, consequentemente, não ocorrem em nenhum dos pares de caminhos compatíveis); agregação de pares de caminhos compatíveis; e agregação de caminhos singulares.

6.2.1 Geração de pares de caminhos

Nesta fase, são analisados todos os pares de caminhos de S e avaliada a sua compatibilidade. Os pares compatíveis serão processados na segunda fase, por uma ordem que envolve três critérios:

1. Ordem decrescente de compatibilidade do par;
2. Ordem decrescente de "potencial de agregação" do par em S;

Listing 6.1: Algoritmo de agregação de caminhos em árvores

```
1  Seja:
2  P - Conjunto de caminhos
3
4  Algoritmo AggregatePaths(P)
5
6  (PathPairs, IndependentPaths) = generatePairs(P);
7  (IndependentPaths, Trees) = aggregatePairs(PathPairs, IndependentPaths);
8  Trees = aggregateIndependentPairs(IndependentPaths, Trees);
9
10 return Trees;
```

3. Ordem decrescente de comprimento do par (que é a soma dos comprimentos dos dois caminhos).

O critério da compatibilidade tem como objetivo privilegiar os pares cuja agregação é a mais "natural", ou seja, aqueles cujo resultado agregado é o menos diferente de cada um dos caminhos. A ordenação pelo potencial de agregação dá prioridade a pares de caminhos que têm um maior número de nós em comum com todos os outros. Formalmente, o *potencial de agregação* de um caminho p no conjunto S é a soma das compatibilidades de todos os pares de caminhos de S que envolvem p:

$$\text{potAgreg}(p, S) = \sum_{q \in S \setminus \{p\}} compat(p, q).$$

O *potencial de agregação* de um par de caminhos (p, q) em S é a soma do potencial de agregação de p e de q em S:

$$\text{potAgreg}((p, q), S) = \text{potAgreg}(p, S) + \text{potAgreg}(q, S).$$

Por fim, com o terceiro critério, pretende-se tratar os caminhos de maior comprimento o mais cedo possível, enquanto existem mais possibilidades de agregação, deixando para o fim os de menor comprimento, que em princípio têm mais facilidade em se agregar.

6.2.2 Agregação dos pares de caminhos

Na fase de agregação dos pares de caminhos compatíveis (esquematizada na listagem 6.2), processa-se um par de cada vez, pela ordem definida acima, até todos os pares terem sido tratados. Começa-se por verificar, para cada caminho do par, se alguma árvore existente o cobre, sendo necessário distinguir três casos:

- Os dois caminhos estão contidos em árvores (possivelmente distintas);
- Nenhum dos caminhos está contido numa árvore;
- Um dos caminhos está contido numa árvore e o outro não está contido em nenhuma árvore.

Listing 6.2: Algoritmo de agregação de caminhos em árvores (Parte de agregação de pares)

```
1  Algoritmo AggregatePairs(PathPairs, IndependentPaths)
2
3  Trees = {};
4
5  for (p1,p2) ∈ PathPairs do \\a permutação de PathPairs realiza-se pela ordem
       determinada em generatePairs
6    if p1 is aggregated p2 and is aggregated
7      ;
8    elsif p1 is not aggregated and p2 is not aggregated
9      tree = findTreeToAggregatePair(p1,p2,Trees);
10     if tree != null
11       aggregatePair(p1,p2,tree);
12     else
13       newTreeFromPair(p1,p2,Trees);
14     end
15   else
16     if p1 is aggregated
17       let t1 be the tree where p1 is aggregated;
18       tryToAggregatePath(p2,IndependentPaths,t1,Trees);
19     else
20       let t2 be the tree where p2 is aggregated;
21       tryToAggregatePath(p1,IndependentPaths,t2,Trees);
22     end
23   end
24 end
```

Se acontecer o primeiro caso, o processamento do par termina imediatamente. No segundo caso, se houver alguma árvore compatível com o par, insere-se o par de caminhos numa das árvores existentes (a escolha desta árvore será detalhada mais à frente); senão, é criada uma nova árvore com este. No último caso (esquematizado na listagem 6.3), em que um dos caminhos está contido numa árvore t e o outro (que designaremos por p) não está contido em nenhuma árvore, se p for compatível com t, o caminho é inserido nessa árvore; se p não for compatível com t mas houver alguma árvore compatível com p, agrega-se p a uma das árvores existentes; e se não existir nenhuma árvore compatível com p, o caminho é adicionado à estrutura dos caminhos singulares (adiando-se a sua agregação).

A procura de uma árvore compatível com um dado caminho ou par de caminhos α devolve, em caso de sucesso, a árvore t onde a inserção de α será efetuada. Essa árvore é, de entre todas as árvores compatíveis com α, uma que maximiza o valor da compatibilidade. Ou seja, se T for o conjunto das árvores existentes, a árvore $t \in T$ onde α é inserido verifica: compat$(\alpha, t) \geq 1$ e $(\forall t' \in T)$ compat$(\alpha, t) \geq$ compat(α, t').

6.2.3 Agregação dos caminhos singulares

Nesta última fase, os caminhos são selecionados por ordem decrescente de comprimento. O processamento de cada caminho p também se inicia com a procura de uma árvore que o contenha, não havendo nada mais a fazer quando esta pesquisa tem sucesso. Se p não

Listing 6.3: Algoritmo de agregação de caminhos em árvores (Função de agregação de um caminho)

```
1  Algoritmo TryToAggregatePath(Path, IndependentPaths, PreferentialTree, Trees)
2
3  if Path is compatible with PreferentialTree
4    aggregatePath(Path,PreferentialTree);
5  else
6    tree = findTreeToAggregatePair(Path,Trees);
7    if tree != null
8      aggregatePath(Path,tree);
9    else
10     addToIndependentPaths(Path,IndependentPaths);
11   end
12 end
```

estiver contido em nenhuma árvore, verifica-se se p é compatível com alguma árvore e, em caso afirmativo, insere-se p numa árvore existente. Quando o caminho é incompatível com todas as árvores, uma nova árvore G_p é criada e adicionada ao conjunto resultado.

6.3 Implementação do algoritmo

A implementação de cada uma das fases será agora analisada com mais detalhe. A complexidade temporal total do algoritmo será depois no final calculada e apresentada.

6.3.1 Geração dos pares caminhos

Na fase da geração de caminhos, a implementação pode ser divida em 2 partes principais: geração dos pares de caminhos e ordenação dos pares de caminhos. Na primeira parte, são percorridas todas as combinações distintas de dois caminhos, $O(|S|^2)$ passos. Para cada combinação de caminhos é calculado o comprimento máximo de uma sequência de nós em comum entre os dois caminhos. Este cálculo é realizado utilizando um vetor de ocorrências com tamanho igual ao número de vértices do grafo. Este vetor é preenchido iterando sobre os nós dos dois caminhos, $O(|V|)$ passos. De seguida, o vetor é percorrido de forma a determinar a compatibilidade entre os caminhos, $O(|V|)$ passos. O cálculo da compatibilidade entre dois caminhos tem portanto complexidade $O(|V|)$, o que faz com que esta primeira parte da fase da geração de pares de caminhos tenha complexidade total $O(|S|^2 \times |V|)$

Na parte da ordenação dos pares de caminhos, são percorridos os pares compatíveis, $O(|S|^2)$ passos no pior caso, sendo cada um adicionado a uma estrutura ordenada (*Java TreeSet*). Esta adição tem complexidade temporal $O(\log(|S|))$. Desta forma, a complexidade da ordenação é $O(|S|^2 \times \log(|S|))$, o que faz com que a complexidade total da fase de geração de pares de caminhos seja $O(|S|^2 \times (|V| + \log|S|))$.

6.3.2 Agregação dos pares de caminhos

Na fase de agregação de caminhos todos os pares compatíveis são percorridos, $O(|S|^2)$ passos.

É verificado previamente se cada um dos caminhos do par já está contido em alguma árvore. Esta verificação tem complexidade $O(|T| \times |S| \times |V|)$, porque o caminho é comparado com todas as árvores já existentes ($O(|T|)$ passos), verificando-se o número de nós em comum com cada uma e a sua compatibilidade com a mesma (apenas é testada a compatibilidade se o número de nós em comum for igual ao número de nós do caminho). O teste de compatibilidade entre o caminho e a árvore é realizado em $O(|S| \times |V|)$ passos (este processo será explicado em maior detalhe mais à frente). Como $|T| \leq |S|$, a complexidade total desta verificação é $O(|S|^2 \times |V|)$.

Após a verificação anterior, o par é tratado por um dos casos apresentados anteriormente. O primeiro caso, em que os dois caminhos do par já se encontram agregados tem complexidade constante, pois o algoritmo passa imediatamente ao par seguinte.

No segundo caso, em que apenas um dos caminhos está agregado, é procurada uma árvore para agregar o outro caminho. Esta procura obriga a percorrer todas as árvores, ou seja, $O(|T|)$ passos. Em cada uma é testada a compatibilidade da árvore com o caminho, $O(|S| \times |V|)$ passos . Se for encontrada uma árvore para agregar o caminho, este é agregado na melhor árvore. Este processo tem complexidade $O(|V|)$ pois todos os nós do caminho são adicionados às estruturas da árvore. Se não for encontrada nenhuma árvore compatível, o caminho é adicionado à estrutura dos singulares, em $\log(|S|)$ passos. Tanto um caso como o outro a complexidade é $O(|T| \times |S| \times |V|)$. Assim, a complexidade final do segundo caso é $O(|T| \times |S| \times |V|)$, ou simplificando, $O(|S|^2 \times |V|)$.

O último caso, em que nenhum dos caminhos se encontra agregado tem complexidade $O(|S| \times |V|)$, pois é feita uma procura semelhante nas árvores existentes, verificando a compatibilidade do par com cada uma, $O(|T| \times |S| \times |V|)$ passos. No final, se existir pelo menos uma árvore compatível, o par é agregado na melhor árvore (das compatíveis). Se não existir, é criada uma nova árvore com o par. Tanto um caso como o outro têm complexidade $O(|V|)$, pois todos os nós do par têm de ser adicionados a uma árvore. Esta complexidade é menor que a da procura por árvores compatíveis, o que faz com que a complexidade final deste caso seja $O(|T| \times |S| \times |V|)$, ou $O(|S|^2 \times |V|)$.

A complexidade final da fase de agregação de pares de caminhos é portanto $O(|S|^4 \times |V|)$.

A compatibilidade entre um caminho e uma árvore pode ser decidida e avaliada em $O(|S| \times |V|)$ passos. Isto deve-se ao facto de inicialmente serem percorridos todos os nós de todos os caminhos já agregados na árvore, de forma a criar estruturas que representem os arcos e nós existentes nesta. Após estes passos iniciais, os nós do caminho são percorridos ($O(|V|)$ passos), de forma a detetar se a adição do caminho à árvore criaria um ciclo. A complexidade total do teste de compatibilidade fica então em $O(|S| \times |V|)$. A compatibilidade entre um par de caminhos e uma árvore também tem complexidade

$O(|S| \times |V|)$. A única diferença entre este teste de compatibilidade e o anterior (com apenas 1 caminho), é a necessidade da verificação de criação de ciclo para os dois caminhos do par, em vez de apenas para um. Este processo pode e deve ser otimizado no trabalho futuro, conseguindo-se com o uso de partições para representar a árvore, uma redução da complexidade do teste de compatibilidade para $O(|V|)$.

6.3.3 Agregação dos caminhos singulares

Na fase de agregação dos caminhos singulares, todos os caminhos que não foram agregados na fase anterior são percorridos, $O(|S|)$ passos. Para cada um, primeiro é verificado se já se encontra agregado em alguma árvore, $O(|S|^2 \times |V|)$ passos. Depois, se não se encontrar em nenhuma árvore, é procurada uma árvore compatível para agregar o caminho, $O(|S|^2 \times |V|)$ passos. Se for encontrada pelo menos uma árvore compatível, o caminho é agregado na melhor ($O(|V|)$ passos), se não, é criada uma nova árvore com o caminho ($O(|V|)$ passos). Esta fase tem portanto uma complexidade total de $O(|S|^3 \times |V|)$.

6.3.4 Complexidade total do algoritmo

A complexidade total da implementação do algoritmo de agregação de caminhos é composta pela soma das complexidades das três fases. Assim, dado que a fase com maior complexidade é a 2ª (fase da agregação dos pares de caminhos), a complexidade final do algoritmo é a desta fase, ou seja, $O(|S|^4 \times |V|)$.

A implementação deste algoritmo ainda pode ser otimizada, não só ao nível da função de teste de compatibilidade, como ao nível do reaproveitamento dos resultados destes mesmos testes. Por exemplo, entre a fase de verificação de se um caminho está contido numa árvore e a fase de procura de uma árvore para agregar o caminho, são realizados alguns dos mesmos testes de compatibilidade, cujos resultados podem ser reaproveitados. Após esta otimização é expectável que a complexidade do algoritmo seja $O(|S|^3 \times |V|)$, no entanto essa versão otimizada ainda não foi implementada.

7

Análise Empírica do Algoritmo de Agregação de Caminhos

O objetivo da agregação de caminhos é diminuir o custo espacial da parametrização do encaminhamento nos nós da rede, tal como foi dito no capítulo anterior. Portanto, quanto menor o custo espacial, melhor será a agregação. Será então neste capítulo feita uma análise comparativa do algoritmo. Isto é, para os mesmos conjuntos de caminhos, serão testados diferentes algoritmos: o nosso e outros algoritmos conhecidos na literatura. Os resultados serão então comparados e analisados.

7.1 Conjuntos de caminhos utilizados

Os conjuntos de caminhos utilizados nos testes serão os mesmos produzidos pelo nosso algoritmo de seleção (apresentado no capítulo 3), tendo sido utilizada, no entanto, para as redes não regulares, a versão com tratamento de exceções ao invés da original:

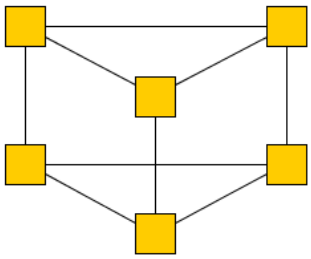
- Full Mesh com os parâmetros k = 5, $h = 1$ e $f = 2$.
- Anel com os parâmetros k = 2, $h = 8$ e $f = 9$.
- Fat Tree com os parâmetros k = 8, $h = 0$ e $f = 1$.
- Folded Clos com os parâmetros k = 6, $h = 0$ e $f = 1$.
- GÉANT com os parâmetros $k = 3$, $h = 5$ e $f = 5$.
- Internet2 com os parâmetros $k = 3$, $h = 5$ e $f = 5$.

- NTT com os parâmetros $k = 3$, $h = 5$ e $f = 5$.
- Aleatória com os parâmetros $k = 3$, $h = 5$ e $f = 6$.

Adicionalmente, foi utilizada uma rede com 6 nós (figura 7.1) usada como exemplo em [Mud+09; Mud+10] (SPAIN), para a qual se conhece o número de árvores ótimo. O conjunto de caminhos para esta rede foi gerado correndo o algoritmo de seleção do SPAIN com $k = 3$. Optou-se por correr este algoritmo em vez do nosso, porque o número ótimo de árvores é apresentado em [Mud+09; Mud+10], sendo válido para o conjunto de caminhos calculados na mesma publicação, ou seja, calculados utilizando algoritmo de seleção SPAIN. O parâmetro $k = 3$ é também o mesmo utilizado nessa publicação.

Para as redes regulares o número de árvores ótimo pode ser calculado através da análise das propriedades da rede e dos caminhos selecionados, sendo portanto conhecido. Para as restantes, não se conhece a solução ótima.

Figura 7.1: Rede apresentada em [Mud+09; Mud+10]



7.2 Algoritmos testados

Os algoritmos de agregação de caminhos em árvores comparados são o nosso e o (primeiro) algoritmo do sistema SPAIN [Mud+09; Mud+10] (referido na secção 2.4). Para os testes os dois foram implementados em Java e todos os testes foram executados num computador com 4 Gbytes de RAM, CPU a 2,53 GHertz e usando a Java VM versão 1.7. Adicionalmente, foi calculado qual o menor número de árvores que poderia ser obtido com a estratégia utilizada em [BGN02] e [SMY00] (referidos na secção 2.4), assumindo que não haveria repetições, e que corresponde a $(n - 1)k$. Esta estratégia será designada por LSPs m-t-p, tendo sido discutida com maior detalhe no capítulo do trabalho relacionado (secção 2.4).

7.3 Resultados dos algoritmos de agregação

Os resultados das execuções dos diferentes algoritmos de agregação para os conjuntos de caminhos escolhidos são apresentados na tabela 7.1. As primeiras quatro colunas da tabela mostram informações sobre o conjunto de caminhos utilizado: rede com a qual ele foi produzido; número de nós de entrada ou saída da rede (n); número de caminhos

Tabela 7.1: Resultados da execução dos algoritmos de agregação.

Redes	n	k	$\|S\|$	# óptimo de árvores	# de subgrafos Nosso	# de subgrafos SPAIN	LSPs m-t-p
Full Mesh	6	5	75	6	6	10	25
Anel	10	2	90	10	10	10	18
Fat Tree	8	2 ou 8	152	8	8	14	56
Folded Clos	6	6	90	6	6	18	30
GÉANT	31	3	1381	—	30	38	90
Internet 2	25	3	898	—	14	16	72
NTT	27	3	1044	—	25	39	78
Aleatória	30	3	1318	—	27	34	87
Rede SPAIN	6	3	45	6	7	6	15

selecionados para cada par de nós (k) e número de caminhos do conjunto. A 5ª coluna mostra o número ótimo de árvores em que cada conjunto poderia ser agregado. Tal como explicado anteriormente, este valor só é conhecido para alguns dos conjuntos. As últimas três colunas apresentam os resultados das diferentes estratégias para cada um dos conjuntos de caminhos.

No caso do algoritmo SPAIN, devido ao facto de ter um cariz aleatório, este foi executado para cada conjunto múltiplas vezes. Ao invés de limitar o número de execuções, limitou-se o tempo utilizado por estas. Isto é, durante um certo intervalo de tempo o SPAIN foi executado repetidamente, sendo escolhida a execução com o menor número de sub-grafos resultante. Nos resultados desta tabela, o intervalo de tempo utilizado corresponde a 5 vezes a duração da execução do nosso algoritmo para o mesmo conjunto. Convém referir que o algoritmo foi implementado como foi descrito na secção 6 e pelos autores [Mud+09; Mud+10] (ver secção 2.4). Por isso, o resultado é um conjunto de grafos acíclicos, não necessariamente conexos (o que justifica o título das duas colunas de resultados na tabela). A oitava coluna da tabela 7.1 apresenta como valores o número de árvores mínimo que seria possível obter com a estratégia LSPs m-t-p para cada conjunto.

Um algoritmo com cariz aleatório pode apresentar em execuções seguidas resultados totalmente díspares, por exemplo, o melhor e o pior. Por este motivo e também devido ao facto do SPAIN ser a solução que mais se aproxima à nossa em termos de resultados, são apresentados na tabela 7.2 os resultados do SPAIN em maior detalhe. Para cada conjunto de caminhos são mostrados os dados da execução deste algoritmo para 5 intervalos de tempo distintos, equivalentes a 1,2,3,4 e 5 vezes o tempo de execução do nosso algoritmo para esse mesmo conjunto. As três colunas do SPAIN na tabela apresentam: o intervalo de tempo no qual o algoritmo foi executado, o número de execuções realizadas durante esse intervalo e o menor número de árvores resultantes em todas as execuções nesse intervalo. Nas colunas do nosso algoritmo é apresentado o tempo original que este demorou a executar para o conjunto de caminhos considerado e o número de árvores resultantes dessa execução.

Tabela 7.2: Resultados detalhados da execução do algoritmo SPAIN.

		SPAIN			Nosso	
Redes	# de vezes o nosso tempo	Tempo de execução	# de execuções	Menor # de sub-grafos	Tempo de execução	# de árvores
Full Mesh	1x	193ms	129	12	193ms	6
	2x	386ms	294	14		
	3x	579ms	422	13		
	4x	772ms	598	15		
	5x	965ms	699	10		
Anel	1x	3ms	88	10	3ms	10
	2x	6ms	91	10		
	3x	9ms	99	10		
	4x	12ms	90	10		
	5x	15ms	83	10		
Fat Tree	1x	2sec 564ms	466	14	2sec 564ms	8
	2x	5sec 128ms	899	16		
	3x	7sec 692ms	1406	15		
	4x	10sec 256ms	1867	15		
	5x	12sec 820ms	2274	14		
Folded Clos	1x	484ms	149	19	484ms	6
	2x	968ms	342	18		
	3x	1sec 452ms	515	16		
	4x	1sec 936ms	554	18		
	5x	2sec 420ms	870	18		
GÉANT	1x	5min 7sec	4021	36	5min 7sec	30
	2x	10min 15sec	8086	39		
	3x	15min 23sec	12131	38		
	4x	20min 31sec	16204	39		
	5x	25min 39sec	20268	38		
Internet2	1x	6min 51sec	6375	16	6min 51sec	14
	2x	13min 43sec	13016	17		
	3x	20min 35sec	19542	17		
	4x	27min 27sec	26067	16		
	5x	34min 19sec	32570	16		
NTT	1x	8min 18sec	5451	38	8min 18sec	25
	2x	16min 36sec	10979	38		
	3x	24min 55sec	16474	40		
	4x	33min 13sec	21951	40		
	5x	41min 32sec	27391	39		
Aleatória	1x	4min 35sec	4503	37	4min 35sec	27
	2x	9min 10sec	8986	39		
	3x	13min 45sec	13540	37		
	4x	18min 21sec	18084	34		
	5x	22min 56sec	22598	34		
Rede SPAIN	1x	26ms	4	8	26ms	7
	2x	52ms	24	7		
	3x	78ms	58	7		
	4x	104ms	113	6		
	5x	130ms	318	6		

Os resultados mostram que o nosso algoritmo para as redes regulares agrega o conjunto de caminhos no número ótimo de árvores. Para a rede SPAIN o nosso algoritmo agrega em mais 1 árvore que o valor ótimo. Para as restantes redes, ao não se saber o valor ótimo, esta avaliação não pode ser realizada. No entanto, analisando os resultados de forma comparativa, repara-se que os do nosso algoritmo são sempre melhores que os dos outros algoritmos. Para a rede Anel, os resultados são iguais para o nosso e para o SPAIN. Nesta rede, existem 10 caminhos maximais que agregam todos os outros e estes são incompatíveis entre si, ou seja, cada um criará a sua própria árvore, não existindo margem para uma melhor ou pior agregação. Os valores da coluna LSPs m-t-p são substancialmente mais elevados do que os resultados obtidos por ambos os algoritmos, indiciando que aquela estratégia de resolução do problema geral não é a mais adequada.

8

Conclusões e trabalho futuro

8.1 Conclusões

Quando se pretende realizar encaminhamento numa rede de forma otimizada ou com grande flexibilidade, o encaminhamento multi-caminho surge como a melhor solução. No entanto, este tem uma complexidade acrescida. Parte desta complexidade deriva da problemática de determinar quais os caminhos a utilizar para encaminhar o tráfego. A escolha de caminhos para a realização de encaminhamento multi-caminho é um problema complexo mas fundamental. A diversidade de critérios, muitas vezes contraditórios, torna bastante difícil a criação de um algoritmo de seleção genérico e independente da estratégia de encaminhamento.

Quando a escolha de caminhos é realizada *a priori*, por não se desejar, ou não se poder, utilizar critérios específicos que condicionem a escolha, o número de caminhos gerados pode ser muito grande, o que pode trazer custos ao nível da complexidade espacial necessária para a sua parametrização nos dispositivos que realizam o encaminhamento. A solução para este problema está na agregação de caminhos, nomeadamente em árvores, de forma a diminuir o número de identificadores distintos necessários para realizar o encaminhamento.

Neste trabalho desenvolvemos dois algoritmos: um algoritmo de seleção de caminhos e um algoritmo de agregação de caminhos em árvores.

O algoritmo de seleção desenvolvido é genérico e independente da estratégia de encaminhamento, parametrizável, e tenta satisfazer os objetivos de tolerância a falhas e distribuição de carga, mantendo e garantindo a qualidade dos caminhos selecionados. Este foi avaliado e testado através de diversas redes sintéticas bem conhecidas, e outras

reais, revelando-se mais abrangente e flexível que outros algoritmos referenciados na literatura para atacar o mesmo problema. Com efeito, trata-se do primeiro algoritmo que tenta sistematicamente acomodar vários tipos de critérios simultaneamente, nomeadamente: não permitir a explosão do número de caminhos selecionados, pois impõe um limite ao número de caminhos distintos selecionados; seleção apenas de caminhos cuja diferença de custo ou comprimento em relação aos caminhos ótimos seja limitada e portanto mais realistas; suporte de tolerância a falhas; e suporte de distribuição de carga e da exploração da capacidade disponível do ponto de vista de cada par de nós.

O algoritmo de agregação desenvolvido segue uma estratégia que privilegia a agregação de caminhos com troços em comum. Os resultados obtidos nos testes realizados são melhores que os dos outros algoritmos recenseados na literatura. Adicionalmente, em todas as redes regulares, o nosso algoritmo agregou os conjuntos de caminhos no número mínimo (ótimo) de árvores.

8.2 Trabalho futuro

Este trabalho insere-se num trabalho mais geral que visa desenvolver algoritmos e mecanismos de engenharia de tráfego que permitam otimizar o funcionamento da rede num quadro de carga variável e desconhecida *a priori*, incluindo a resposta atempada às falhas dos canais e dos nós. No futuro, os caminhos e as árvores calculadas poderão ser avaliadas em termos da sua adequação à engenharia de tráfego e à tolerância a falhas. Adicionalmente, este trabalho poderá servir de base para o estudo de novos algoritmos e mecanismos de encaminhamento multi-caminho.

Especificamente, ao nível do algoritmo de seleção de caminhos, aproveitando a sua capacidade de análise, no futuro este poderá evoluir para um mais inteligente que adaptaria dinamicamente os valores de h e f em função dos resultados obtidos para cada par de nós. Por exemplo, estes parâmetros poderiam funcionar como valores base, que se necessário e de forma a atingir o conjunto de caminhos com as características e a cardinalidade pretendidas, seriam dinamicamente alterados consoante o par de nós. O parâmetro k funcionaria então como um objetivo, em que para o atingir, se necessário, seriam ignorados os limites impostos por h e f. Adicionalmente, de forma a melhorar o tempo de execução total do algoritmo para todos os pares de uma rede, este poderá ser paralelizado.

No caso do algoritmo de agregação, tal como referido na secção 6, este pode ser otimizado ao nível da sua implementação. Por exemplo, poderão ser usadas partições para representar os caminhos e as árvores, facilitando a sua comparação em diversas funções do algoritmo, como o cálculo da compatibilidade e a verificação de se um caminho está contido numa árvore. Ao nível da análise do algoritmo, poderá ser interessante a realização de um estudo que deduza a distância máxima das soluções por ele computadas ao valor ótimo.

Bibliografia

[BGN02] S. Bhatnagar, S. Ganguly e B. Nath. "Label space reduction in multipoint-to-point LSPs for traffic engineering". Em: *Universal Multiservice Networks, 2002. ECUMN 2002. 2nd European Conference on* (2002), pp. 29–35.

[BGN03] S. Bhatnagar, S. Ganguly e B. Nath. "Creating multipoint-to-point LSPs for traffic engineering". Em: *Workshop on High Performance Switching and Routing, 2003, HPSR.* (2003), pp. 201–207.

[CCK10] M. Caesar, M. Casado e T. Koponen. "Dynamic route recomputation considered harmful". Em: *ACM SIGCOMM Computer Communication Review* 40.2 (2010), pp. 66–71.

[Cal90] R. Callon. "Use of OSI IS-IS for routing in TCP/IP and dual environments". Em: *IETF, RFC 1195* (1990).

[Cor+09] T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein. *Introduction to Algorithms*. Third. The MIT Press, 2009, p. 1312.

[Dij59] E. Dijkstra. "A note on two problems in connection with graphs". Em: *Numerische Mathematik* 1.269-270 (1959), p. 6.

[FT00] B. Fortz e M. Thorup. "Internet traffic engineering by optimizing OSPF weights". Em: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* 2 (2000), pp. 519–528.

[Fra+05] P. Francois, C. Filsfils, J. Evans e O. Bonaventure. "Achieving sub-second IGP convergence in large IP networks". Em: *SIGCOMM Comput. Commun. Rev.* 35.3 (jul. de 2005), pp. 35–44.

[Hec06] O. M. Heckmann. *The Competitive Internet Service Provider*. 1ª ed. Wiley Series in Communications Networking & Distributed Systems. Chichester, UK: Wiley-Interscience, 2006.

[Hed88] C. Hedrick. "Routing information protocol". Em: *IETF, RFC 1058* (1988).

[Hop00] C. Hopps. "Analysis of an Equal-Cost Multi-Path Algorithm". Em: *IETF, RFC 2992* (2000).

[HMM13a] J. Horta, M. Mamede e J. L. Martins. "Encaminhamento multi-caminho baseado num número reduzido de árvores". Em: *Atas da 13ª Conferência sobre Redes de Computadores*. Nov. de 2013, pp. 103–108.

[HMM13b] J. Horta, M. Mamede e J. L. Martins. "Selecção de caminhos para Encaminhamento Multi-Caminho". Em: *Actas do Inforum 2013 - Simpósio de Informática*. Set. de 2013, pp. 78–89.

[JT92] T. R. Jensen e B. Toft. *Graph Coloring Problems*. Wiley Series in Discrete Mathematics and Optimization. Wiley InterScience, 1992.

[LC02] G. Lee e J. Choi. "A survey of multipath routing for traffic engineering". Em: *Information and Communications University, Korea* (2002).

[Lee+02] Y. Lee, Y. Seok, Y. Choi e C. Kim. "A constrained multipath traffic engineering scheme for MPLS networks". Em: *Communications, 2002. ICC 2002. IEEE International Conference on*. Vol. 4. IEEE, 2002, pp. 2431–2436. ISBN: 0-7803-7400-2.

[Moy97] J. Moy. "OSPF version 2". Em: *IETF, RFC 1247* (1997).

[Mud+09] J. Mudigonda, P. Yalagandula, M. Al-Fares e J. Mogul. *SPAIN: Design and algorithms for constructing large data-center ethernets from commodity switches*. Rel. téc. Tech. Rep. HPL-2009-241, HP Labs, 2009.

[Mud+10] J. Mudigonda, P. Yalagandula, M. Al-Fares e J. Mogul. "Spain: Cots data-center ethernet for multipathing over arbitrary topologies". Em: *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association. 2010, pp. 18–18.

[NST99] P. Narvaez, K.-Y. Siu e H.-Y. Tzeng. "Efficient Algorithms for Multi-Path Link-State Routing". Em: *ISCOM'99, Kaohsiung, Taiwan* (1999).

[NZ01] S. Nelakuditi e Z.-L. Zhang. "On Selection of Paths for Multipath Routing". Em: *Quality of Service — IWQoS 2001*. Ed. por L. Wolf, D. Hutchison e R. Steinmetz. Vol. 2092. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 170–184.

[Per85] R. Perlman. "An algorithm for distributed computation of a spanningtree in an extended LAN". Em: *ACM SIGCOMM Computer Communication Review* 15.4 (set. de 1985), pp. 44–53.

[RL95] Y. Rekhter e T. Li. "A border gateway protocol 4 (BGP-4)". Em: *IETF, RFC 1771* (1995).

[RVC01] E. Rosen, A. Viswanathan e R. Callon. "Multiprotocol label switching architecture". Em: *IETF, RFC 3031* (2001).

[SMY00] H. Saito, Y. Miyao e M. Yoshida. "Traffic engineering using multiple multipoint-to-point LSPs". Em: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. 2000, 894–901 vol.2.

[SN02] G. Schneider e T. Nemeth. "A simulation study of the OSPF-OMP routing algorithm". Em: *Computer Networks* 39.4 (2002), pp. 457–468. ISSN: 1389-1286.

[SFM08] F. Solano, R. Fabregat e J. Marzo. "On optimal computation of MPLS label binding for multipoint-to-point connections". Em: *Communications, IEEE Transactions on* 56.7 (jul. de 2008), pp. 1056–1059.

[Suc+11] M. Suchara, D. Xu, R. Doverspike, D. Johnson e J. Rexford. "Network architecture for joint failure recovery and traffic engineering". Em: *SIGMETRICS Performance Evaluation Review-Measurement and Evaluation* 39.1 (2011), p. 97.

Lista de abreviaturas

BGP Border Gateway Protocol

CPU Central Processing Unit

ECMP Equal-Cost Multipath

HTTP Hypertext Transfer Protocol

IGP Interior Gateway Protocol

IP Internet Protocol

ISP Internet Service Provider

IS-IS Intermediate System to Intermediate System

LER Label Edge Router

LSA Link State Anouncement

LSP Label Switched Path

LSR Label Switching Router

MPA Multiple path Algorithm

MPLS Multiprotocol Label Switching

m-t-p Multipoint-to-Point

NTT Nippon Telegraph and Telephone

OSPF Open Shortest Path First

OSPF-OMP OSPF Optimized Multipath

RAM Random-Access Memory

RIP Routing Information Protocol

SLA Service Level Agreement

STP Spanning Tree Protocol

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

ToR Top of Rack

TTL Time to Leave

VLAN Virtual Local Area Network

VoIP Voice over IP