Fábio André Cardoso da Silva

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Um estudo de algoritmos de gestão de tráfego de baixa complexidade em SDNs

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Pedro Miguel Figueiredo Amaral, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente:	Prof. Doutor José Manuel Matos Ribeiro da Fonseca
Arguente:	Prof. Doutor Luís Filipe Lourenço Bernardo
Vogal:	Prof. Doutor Pedro Miguel Figueiredo Amaral

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2016

Um estudo de algoritmos de gestão de tráfego de baixa complexidade em SDNs

To my long-time companion, Cláudia.

Agradecimentos

Em primeiro lugar quero deixar um obrigado especial a todos os professores do Departamento de Engenharia Eletrotécnica da Faculdade de Ciências e Tecnologia que contribuíram para a minha formação, pois sem eles hoje não estaria aqui.

Quero agradecer ao meu orientador, Prof. Pedro Amaral, por toda a orientação cedida, ajuda, paciência e acima de tudo disponibilidade. A porta do seu gabinete esteve sempre aberta para as longas conversas que realizamos e que contribuíram para o desenvolvimento desta tese.

A todos os meus colegas que me acompanharam, o meu obrigado, em especial ao André Estevam, Daniela Oliveira, David Mestre, Duarte Segurado, Fábio Carmo, Gonçalo Freitas, Joana Barruncho e Miguel Prego. A camaradagem, amizade, gargalhadas e todos os bons momentos nunca serão esquecidos. Foram com certeza dos melhores momentos que vivi até hoje.

Aos meus pais e ao meu irmão o meu gigante obrigado por todo o suporte diário e compreensão do esforço necessário para realizar o curso.

Em último lugar quero agradecer à minha namorada, Cláudia Fonseca, que nestes últimos 5 anos com a sua clareza, calma e inteligência ajudou-me a ultrapassar todos os desafios que me apareceram. É sem dúvida a minha cara-metade e sem ela nada disto teria sido possível.

Resumo

A falta de soluções que consigam responder às necessidades das redes atuais gera uma má gestão do tráfego que origina situações de congestionamento quando ainda existem partes da rede livres. Nas redes tradicionais cada nó apenas consegue determinar qual o próximo nó para qual o tráfego deve ser encaminhado, esta característica aliada à falta de conhecimento global da rede torna a tarefa de otimizar o custo das ligações difícil, sendo complexo a obtenção de uma solução ótima.

O uso de *Software-Defined Networking* traz vantagens que permitem contornar as adversidades das redes tradicionais, tornando-se possível desenhar novos métodos de encaminhamento de tráfego, mais eficientes e que considerem o estado global da rede. Aliar estas vantagens ao uso de novos métodos de encaminhamento poderá traduzir-se numa melhoria do desempenho das redes, como por exemplo conseguir um aumento na eficiência no transporte de dados.

A separação do plano de controlo resulta em algumas condicionantes de escalabilidade que condicionam a utilização de métodos de distribuição de tráfego baseados em otimização. As duas condicionantes principais são o número de regras suportadas pelos equipamentos e o custo da comunicação entre os equipamentos na rede e o controlador. Este trabalho parte de uma arquitetura que tem essas limitações em conta, onde a eficiência da utilização da rede depende de dois fatores: o cálculo de caminhos na rede e a distribuição de tráfego por esses mesmos caminhos.

Nesta dissertação são estudadas novas abordagens aos métodos de cálculo de caminhos entre os nós de acesso e aos métodos de distribuição de tráfego tradicionais, focando-se no desenvolvimento de uma ferramenta de modelação teórica que permite estudar as várias alternativas aos métodos de encaminhamento tradicionais fornecendo um indicador de desempenho referente ao método estudado. Foram realizadas várias experiências que demonstram o impacto no transporte de dados de diferentes métodos de encaminhamento.

Palavras-chave: engenharia de tráfego, distribuição de tráfego, SDN, OpenFlow, encaminhamento algébrico, grafos.

Abstract

The lack of solutions that can answer to current network needs generates poor management of traffic that leads to congestion when there are still free parts in the network. In traditional networks each node can only determine the next node to which traffic should be routed, this feature combined with the lack of global knowledge of the network makes the task of optimizing the cost of links hard, being complex to obtain an optimal solution.

The use of emph Software-Defined Networking brings advantages that allow to overcome the adversities of traditional networks, making it possible to design new, more efficient traffic routing methods that consider the overall state of the network. Combining these advantages the use of new routing methods could result in improved network performance, as for example to achieve an increased efficiency in data transport.

The separation of the control plane scalability results in some conditions which restrain the use of optimization-based traffic distribution methods. The two main constraints are the number of rules supported by the equipment and the cost of communication between the devices on the network and controller. This work starts from an architecture that has these limitations into account, where the efficiency network utilization depends on two factors: the calculation of the paths in the network and the distribution of traffic by these same paths.

This thesis will study new approaches regarding path calculation methods between access nodes and traditional traffic distribution methods, focusing on the development of a theoretical modeling tool to study the various alternatives to traditional routing methods providing a performance indicator for the studied method. Several experiments demonstrating the impact on data transport of different routing methods were carried out.

Keywords: traffic engineering, traffic distribution, SDN, OpenFlow, algebric routing, graphs.

Índice

Lista de Figuras

Lista de Tabelas

Siglas

APIs *Application Programming Interfaces*.

BFS *Breadth-First Search*.

ECMP *Equal-cost multi-path*.

LLDP *Link Layer Discovery Protocol*.

MCF *Multi-commodity flow*.

MPLS *Multiprotocol Label Switching*.

MPLS-TE *Multiprotocol Label Switching-Traffic Engineering*.

PL Programação Linear.

QoS *Quality of Service*.

SDN *Software-Defined Networking*.

SDN-FE Elemento de Encaminhamento SDN.

SDN-C Controlador SDN.

TCP *Transport Control Protocol*.

TE *Traffic Engineering*.

TLS *Transport Layer Security*.

WAN *Wide Area Network*.

Capítulo 1

Introdução

1.1 Motivação

Como consequência da informatização do mundo, a utilização das redes de computadores tem crescido a um ritmo elevado. As redes atuais têm de suportar um conjunto de aplicações e usos para os quais os protocolos e soluções técnicas atuais, muitas vezes, não encontram resposta. Um exemplo é a falta de eficiência dos métodos de encaminhamento de tráfego que foram desenhados para funcionarem de forma fiável e distribuída num cenário em que a quantidade de tráfego a transportar era muito inferior.

Uma maneira de analisar os problemas de encaminhamento é considerar a rede como um grafo, onde os vértices são os nós constituintes da rede e as arestas são as ligações entre os nós. Considerando a abstração referida, a forma mais comum de encaminhamento é o problema do caminho mais curto, *shortest path*. Este problema consiste em encontrar um caminho entre dois vértices, de forma a que o custo do caminho seja minimizado. Em encaminhamento multi-caminho o problema do caminho mais curto é referido como *Equal-cost multi-path* (ECMP), que consiste numa estratégia de encaminhamento onde os pacotes entre dois vértices são encaminhados por vários caminhos com o mesmo custo. O custo de um caminho é geralmente uma métrica numérica, tais como, a largura de banda disponível no mesmo, o numero de *hops*, isto é, o número de nós que distancia os vértices que têm uma ligação ativa, a latência da ligação, entre outras.

As redes atuais, de uma maneira geral, usam uma abordagem de encaminhamento de forma a resolver o problema do caminho mais curto. Apesar de simples e escalável, a utilização de ECMP e de informação local leva a uma gestão do tráfego de uma maneira estática gerando a uma má utilização dos recursos disponíveis da rede, sendo possível a ocorrência de congestionamentos em determinados pontos da rede quando existem recursos disponíveis. A falta de conhecimento global e a incapacidade de determinar

mais do que qual o próximo nó para encaminhar tráfego, tornam a tarefa de otimizar o custo das ligações difícil, o que reflete a complexidade em obter uma distribuição ótima de encaminhamento.

Software-Defined Networking (SDN) é um novo paradigma de redes que pode mudar este cenário, introduzindo um nível maior de programação e flexibilidade à rede. Em SDN o plano de dados é separado do plano de controlo, deixando de existir a verticalização das redes tradicionais. Neste contexto, o plano de controlo passa para um dispositivo da rede que detêm uma visão global da rede, o controlador, facilitando a implementação de políticas de rede e de novos protocolos.

A utilização de centralização total no controlador traz outros desafios, nomeadamente ao nível da escalabilidade, dificultando a otimização da distribuição do tráfego. Contudo, a programação da rede pode ser explorada para estudar o uso de outras aproximações de encaminhamento para construir mecanismos escaláveis, que ainda assim permitem aumentar a eficiência da utilização dos recursos da rede de modo pouco complexo.

O objetivo desta dissertação é o estudo de métodos de encaminhamento e de distribuição de tráfego que possam ser utilizadas na arquitectura definida em [1]. Neste trabalho foi definida uma arquitetura que tira partido das vantagens das SDN, nomeadamente, a nível de programação da rede e visão global, com o intuito de obter uma gestão de tráfego de modo escalável. Nesta é utilizada uma lógica de controlo hierárquica, dividida em dois níveis, como abordagem aos problemas tradicionais de escalabilidade, nomeadamente o número de regras implementáveis nos nós de encaminhamento e o custo da comunicação com o controlador.

O comportamento da arquitetura foi modelado de forma teórica de modo a utilizar um software que calcula o tráfego possível de transportar numa topologia, para diversas combinações de algoritmos de cálculo de caminhos e distribuição de tráfego pelos mesmos. Foram depois estudados vários tipos de encaminhamento que podem ser utilizados na arquitetura, mantendo a simplicidade de encaminhamento *hop-by-hop* baseado no destino e a utilização de algoritmos de baixa complexidade. Foi também estudada a forma como se distribui o tráfego por entre os vários caminhos igualmente "bons"através de métricas de grafos como a centralidade ou outras. Finalmente, foram realizadas simulações utilizando *switches* virtuais OpenFlow controlados por um controlador SDN.

1.2 Organização da Dissertação

Para além deste capítulo de introdução, esta dissertação está organizada em mais quatro capítulos:

- **Capítulo 2 - Trabalho Relacionado**
  Neste capítulo é apresentado o estado da arte dos temas abordados nesta dissertação. Começa-se por descrever soluções tradicionais de *traffic engineering*, explicando o porquê de não conseguiram satisfazer os requisitos das redes atuais. De seguida é

realizada uma descrição teórica de um novo paradigma de redes de computadores conhecido como *software-defined networking*, onde se pode tirar partido das enumeras vantagens para realizar uma distribuição de tráfego que satisfaça os requisitos atuais das redes. O uso de SDN permite desenvolver soluções de *traffic engineering* interessantes, contudo é necessário o uso de uma arquitetura que permita contornar os problemas de escalabilidade introduzidos. Neste capítulo descreve-se a arquitetura de controlo hierárquica de dois níveis utilizada, explicando como soluciona estes problemas. Finaliza-se o capítulo descrevendo como é que se pode utilizar uma álgebra para o cálculo dos caminhos entre os nós da rede e qual o impacto que poderá ter no transporte de dados face a soluções tradicionais.

- **Capítulo 3 - Descrição do Projeto**
  O capítulo 3 é o capítulo onde é descrito o trabalho elaborado. Descreve-se a implementação de uma ferramenta de modelação teórica que permite o estudo de vários métodos de encaminhamento e a implementação de uma aplicação SDN que permite validar os resultados teóricos. São apresentadas alternativas aos métodos tradicionais de cálculo dos caminhos e de cálculo da distribuição de tráfego.

- **Capítulo 4 - Simulações**
  No quarto capítulo são expostos os resultados obtidos através da ferramenta de modelação teórica e da aplicação SDN. Começa-se por descrever os ambientes de simulação em termos de tecnologia utilizada, algoritmos de simulação e cálculo do pior cenário de teste. De seguida apresenta-se os resultados acompanhados de justificações para os mesmos.

- **Capítulo 5 - Conclusões e Trabalho Futuro**
  Finaliza-se comentado os resultados obtidos e referindo as conclusões que se podem tirar do desenvolvimento deste trabalho. Apresenta-se sugestões de trabalho futuro relacionadas com esta dissertação.

1.3 Contribuições Principais

A contribuição principal desta dissertação foi a implementação de uma ferramenta de modelação teórica, que através de uma matriz de adjacência permite o estudo de forma analítica de redes, assim como o estudo de vários métodos de encaminhamento, nomeadamente o cálculo dos caminhos e distribuição do tráfego. Em adição, calcula o conjunto de nós que quando a transmitir ao mesmo ritmo provocam uma situação de congestionamento na rede mais rapidamente e realiza uma simulação teórica que serve como indicador do desempenho dos diferentes mecanismos de encaminhamento.

Através da aplicação desenvolvida que valida os resultados teóricos obtidos é possível estudar o impacto dos vários métodos estudados e retirar conclusões que permitam desenvolver novas soluções de encaminhamento como alternativa a soluções clássicas.

Capítulo 2

Trabalho Relacionado

Neste capítulo, começa-se por apresentar, resumidamente, os mecanismos de encaminhamento mais utilizados em redes tradicionais, sendo também apresentados os mecanismos de *Traffic Engineering* (TE) utilizados, referindo-se os problemas que existem na utilização destes mecanismos. É introduzida a otimização *multi-commodity* como a solução clássica para obter utilização ótima, referindo os problemas existentes na sua aplicação em redes tradicionais.

Na secção seguinte introduz-se uma nova arquitetura em redes de computadores, conhecida como SDN, salientado-se as suas vantagens e limitações em relação ao problema da utilização da capacidade da rede. São apresentadas algumas aproximações existentes para SDN de modo a compreender as limitações atuais, nomeadamente os problemas de escalabilidade.

Por fim, apresentam-se as bases necessárias e os conceitos mais importantes para compreender de que forma se pode utilizar álgebra para modelar o comportamento de um protocolo de encaminhamento e como estes modelos podem servir para desenhar protocolos que levem a uma maior eficácia na rede. São apresentadas as condições necessárias para garantir o correto funcionamento do protocolo.

2.1 Redes Tradicionais

Nas redes tradicionais o encaminhamento de tráfego é efetuado de acordo com o resultado da aplicação de um algoritmo de controlo distribuído responsável pela recolha da informação necessária para que os nós constituintes das redes consigam tomar decisões de encaminhamento face aos pacotes recebidos. Neste processo cada nó consegue apenas definir o próximo nó do caminho.

A eficiência na utilização dos recursos da rede está ligada ao conceito de *traffic engineering* que pode ser visto, de uma maneira geral, como a distribuição do tráfego que circula na rede de forma a atingir um certo objetivo. Trata-se de certa forma de uma otimização do encaminhamento do tráfego na rede.

Nas próximas secções são discutidos os métodos mais comuns de TE nas redes atuais.

2.1.1 TE baseado em *Multiprotocol Label Switching*

Multiprotocol Label Switching (MPLS) [2] é uma técnica de encaminhamento muito utilizada em redes de computadores de alto desempenho. O encaminhamento em MPLS consiste em encaminhar os pacotes por caminhos predefinidos denominados *túneis*, que são identificados por *labels*. O uso de *labels* para identificar os caminhos em que os pacotes devem circular simplifica as pesquisas nas tabelas de encaminhamento dos nós.

Multiprotocol Label Switching-Traffic Engineering (MPLS-TE) é um dos mecanismos de TE mais utilizados em redes de computadores sendo suportado pela maior parte dos vendedores de *routers* [3]. De uma maneira resumida, o MPLS-TE consiste na utilização de um algoritmo que encontra o caminho mais curto entre dois nós que consiga a acomodar o tráfego estimado, que irá ser transmitido no *túnel*, realizando uma reserva de recursos em todas as ligações[1] do caminho escolhido que é utilizada no cálculo de novos caminhos, de modo a conhecer os recursos ainda disponíveis.

Contudo o uso de MPLS-TE não produz a eficiência desejada na utilização dos recursos. No trabalho realizado em [3] é abordado o problema da inflação da latência no uso de MPLS-TE numa *Wide Area Network* (WAN) que interliga *data centers* da Microsoft, referida como MSN. Nenhum nó tem conhecimento global da rede e os *routers* geradores de tráfego escolhem os *túneis* que satisfaçam os seus requisitos de tráfego atuais, causando assim problemas de latência quando a mudança de carga excede a capacidade de um subconjunto de ligações ao longo dos caminhos mais curtos utilizados pelos *túneis* [3], [4]. O MPLS-TE usa um algoritmo de otimização que procura sempre o melhor caminho para a distribuição do tráfego, sendo que a inflação da latência ocorre quando os requisitos de tráfego mudam e o algoritmo designa outro caminho para a distribuição do tráfego [3].

Os requisitos de tráfego numa rede como a anterior variam bastante, pois os serviços enviam diferentes quantidades de tráfego sem considerar o estado da rede e o tráfego de outros serviços. Esta falta de coordenação resulta na oscilação da rede entre um estado de má alocação de recursos e um estado de alocação de recursos em excesso [4]. Uma forma comum de aprovisionamento é alocar recursos com base na carga máxima que irá circular na rede, contudo a utilização média das ligações seria de 50% [4], confirmando a falta de eficiência de algoritmos de TE baseados em MPLS na distribuição do tráfego.

[1] O termo ligação é utilizado para definir a ligação direta entre dois nós, o termo caminho é utilizado para definir um conjunto de ligações que definem a rota pela qual os pacotes circulam entre dois nós.

2.1.2 TE baseado em Otimização

Nas redes tradicionais, os recursos necessários para satisfazer os requisitos de tráfego são alocados por cada nó numa decisão local baseada nos caminhos calculados por algoritmos distribuídos que visam a resolver o problema do *caminho mais curto*, onde a seleção do caminho é baseada nos custos atribuídos às várias ligações. Cada nó apenas consegue controlar qual o próximo nó onde irá circular o tráfego e apenas tem conhecimento local da rede. Esta característica das redes tradicionais traduz-se numa má utilização dos recursos da rede, sendo que existem ligações congestionadas quando existem partes da rede que estão livres [1].

Desta forma, a única maneira de se conseguir melhorar o desempenho da rede e influenciar a distribuição do tráfego é otimizar os custos das várias ligações de modo a obter um conjunto de caminhos que maximize a utilização dos recursos da rede. Este problema denomina-se *NP-hard* [5]. A forma geral do problema da otimização da distribuição do tráfego numa rede é conhecido como *Multi-commodity flow* (MCF), que é resolvido por Programação Linear (PL) [1].

No trabalho realizado em [6] demonstra-se que é possível obter os pesos ótimos num protocolo *Link-State* com encaminhamento *hop-by-hop* baseado no destino, sem ter que se lidar com a complexidade do problema de MCF. De referir ainda que a otimização vem à custa de se distribuir tráfego por caminhos que não são os mais curtos. É necessário utilizar uma abordagem de encaminhamento multi-caminho, onde o tráfego é encaminhado por vários caminhos alternativos. A implementação mais conhecida de encaminhamento multi-caminho é o ECMP e nas redes tradicionais, de uma forma geral, apenas se consegue utilizar os caminhos mais curtos tornando a solução apresentada em [6] pouco prática. Em [7] é apresentado o HALO, o primeiro algoritmo de encaminhamento que consegue obter os pesos ótimos de forma a resolver o problema de MCF. De forma semelhante descrita em [6], o algoritmo mantém a simplicidade de encaminhamento *hop-by-hop* baseado no destino, contudo o tempo de convergência do algoritmo pode ser um problema e depende de atualizações síncronas entre os *routers*. A implementação do HALO é difícil visto ser necessário alterar o plano de controlo dos *routers*, o que implica atualizar todos os nós da rede, uma vez que nas redes tradicionais o plano de controlo é executado nos *routers*.

O uso de MPLS e, por consequente de *explicit routing*, simplifica o problema da otimização, ainda que a otimização seja obtida à custa do estabelecimento de vários túneis entre os nós que podem ter de se alterar consoante os requisitos de tráfego [1], [6]. Usando MPLS é possível encontrar os pesos ótimos de uma maneira simples, se estivermos a resolver o problema de MCF *offline*. No entanto o problema torna-se muito mais complicado *online* devido às variações dos requisitos de tráfego e aos problemas de escalabilidade [3].

2.2 Software-Defined Networking

O novo paradigma introduzido pelas SDN simplifica o problema da eficiência do transporte de dados nas redes, abrindo novas oportunidades nesta área. A separação do plano de controlo do plano de dados introduz um nível maior de programação e flexibilidade face às redes tradicionais. O plano de controlo é implementado num dispositivo de software que detém uma visão global da rede, permitindo decisões de encaminhamento que têm em consideração o estado global da rede, contrariamente às redes tradicionais, onde as decisões de encaminhamento são efetuadas com base em informação local.

2.2.1 Arquitetura SDN

Software-Defined Networking [8] é um paradigma de redes emergente que visa ultrapassar algumas das limitações das redes tradicionais [9]. Nestas o protocolo de encaminhamento é executado no nó que faz o encaminhamento do pacote, ou seja, o plano de controlo (que é responsável por decidir quais as ações a realizar com o tráfego que circula na rede) e o plano de dados (que encaminha o tráfego de acordo com as decisões do plano de controlo) estão juntos no dispositivo de rede. A ideia principal da arquitetura SDN consiste em separar o plano de controlo do plano de dados fornecendo capacidade de programação no plano de controlo [10]. Esta separação do plano de controlo e do plano de dados tem sido motivo de bastante interesse científico devido aos grandes benefícios operacionais.

As SDNs são constituidas por dois componentes principais [11]:

- **Controlador SDN (SDN-C):** O controlador SDN é o componente responsável pelo o plano de controlo das redes de SDN. É uma função logicamente centralizada que determina o caminho pelo qual o tráfego deve circular. O SDN-C é um bloco de software.
- **Elemento de Encaminhamento SDN (SDN-FE):** Os elementos de encaminhamento SDN constituem o plano de dados. São responsáveis pelo encaminhamento do tráfego, seguindo indicações do SDN-C. Os SDN-FE são componentes de hardware, contrariamente ao SDN-C.

Ao contrário das redes tradicionais, as SDN são redes onde o controlo se encontra separado do plano de encaminhamento, sendo que o controlador SDN comunica com todos os SDN-FE, como é exemplificado na figura 2.1. A comunicação entre o controlador e os SDN-FE é realizada através do protocolo OpenFlow [12], o protocolo aceite como padrão na comunidade.

A possibilidade de centralizar o controlo começou por levantar alguns problemas na comunidade, principalmente, face à escalabilidade e segurança das SDN, sendo estes dois dos problemas que as SDN ainda têm por resolver [9], [10], [13], [14] . Os problemas induzidos pela centralização do controlo nas SDN é um tópico de bastante interesse, sendo alvo de muita investigação na comunidade científica com o intuito de serem solucionados.

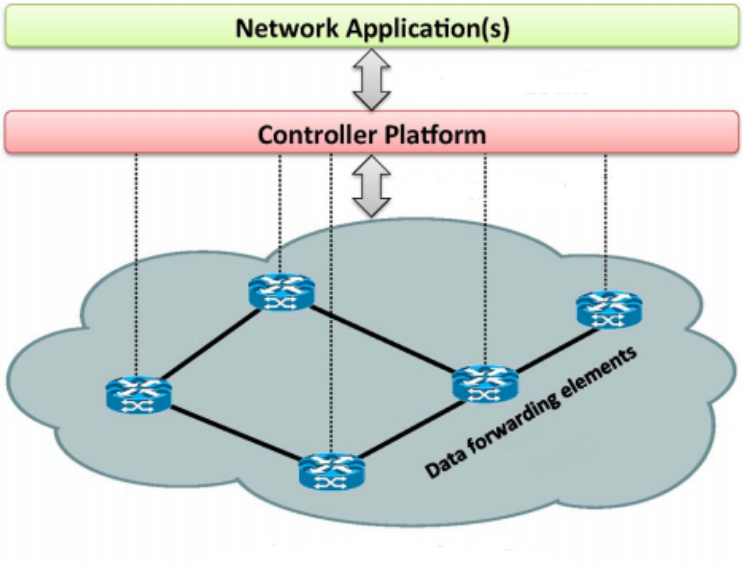

Figura 2.1: Arquitetura simplificada de uma SDN. (Adaptado de [9])

Contudo a separação do controlo torna a rede altamente flexível [15], trazendo bastantes benefícios face às redes tradicionais, ao nível da flexibilidade, agilidade e eficiência. Face a estas vantagens e ao facto de existirem cenários, como as redes de *data centers* que suportam serviços *cloud* [10], onde as redes tradicionais não conseguiram dar resposta levaram a um grande interesse e esforço de padronização e adoção das SDN.

2.2.1.1 Plano de Controlo

É no plano de controlo que reside toda a inteligência da rede, sendo o controlador SDN responsável por controlar a rede sem que seja necessário controlar cada dispositivo de forma individual [14] como acontece nas redes tradicionais. O fato do plano de controlo ser implementado num módulo de software é que permite esta característica às SDN.

Apesar do SDN-C poder constituir um eventual ponto de estrangulamento é importante referir que o modelo de separação de controlo não significa que seja usado apenas um controlador. Na realidade para se garantir níveis adequados de desempenho, escalabilidade e robustez, pode ser adotada uma solução com vários controladores [9]. Um projeto muito conhecido que visa fornecer uma plataforma de controlo distribuída para redes de produção de alta escala é o *Onix* [16].

Duas das características mais importantes do controlador são a rapidez com que consegue responder a pedidos de caminho por parte dos SDN-FE, e quantos desses pedidos consegue cada um processar por segundo [14]. Existem vários controladores SDN que permitem o desenvolvimento de aplicações de controlo para a rede. Destes destacam -se alguns dos mais usados e com mais impacto nas SDN.

O primeiro controlador OpenFlow a ser implementado foi o NOX [17], que está escrito

em C++. Do NOX surgiram dois outros controladores o POX [18], que está implementado em Python, e o controlador NOX-MT [19], que surge como um controlador multi-threaded ligeiramente modificado face ao NOX. O controlador Beacon [20] é implementado em Java e evoluiu posteriormente para o controlador Floodlight [21]. O Floodlight é um projeto *open source* suportado por um fabricante (Big Switch Networks) que tem uma estrutura por módulos assente numa SDK Java que representa o protocolo OpenFlow. É possível consultar uma lista extensiva dos controladores que existem e quem são os respetivos fabricantes em [22].

Com o protocolo assinado entre a Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, a Reditus e a Hewlett Packard Portugal (HP), a HP forneceu equipamento SDN para a realização de investigação e de projetos na área das redes SDN. Consequentemente o controlador HP VAN SDN, implementado em Java e concebido para operar em redes de produção é o utilizado nesta dissertação.

2.2.2 Interfaces de Comunicação

Na figura 2.1 é apresentada uma arquitetura simplista de uma SDN com três componentes: aplicações, controlador e SDN-FEs. A comunicação entre eles é realizada através de um conjunto de *Application Programming Interfaces* (APIs).

Definem-se então duas interfaces responsáveis pela troca de informação dos componentes de uma SDN [13]:

- ***Northbound* API:** Permite a troca de dados entre o controlador SDN e as aplicações que controlam a rede. O tipo de informação que é trocado por esta API depende das aplicações, como tal não existe nenhum protocolo padronizado nesta interface.

- ***Southbound* API:** É a interface responsável pela troca de informação entre o controlador SDN e os SDN-FE. É esta interface que possibilita a separação do plano de controlo do plano de encaminhamento de dados. Ao contrário da *Northbound* API, nesta interface existe um protocolo padronizado, o OpenFlow [12].

Na figura 2.2 é apresentado um diagrama que contém as APIs que possibilitam a interação das diferentes camadas de uma SDN, assim como a direção da comunicação efetuada.

Com as decisões de encaminhamento a serem enviadas para os SDN-FE, os elementos que constituem o plano de encaminhamento de dados numa SDN passam a ser dispositivos de encaminhamento simples, sem controlo embutido e sem a necessidade de tomarem decisões autónomas, como acontecia nas redes tradicionais.

A existência da *Southbound* API possibilita a reprogramação dos SDN-FE de forma dinâmica e heterogénea, algo que era bastante difícil em redes tradicionais devido à existência de grande variedade de interfaces proprietárias fechadas e devido à natureza distribuída do plano de controlo [9].

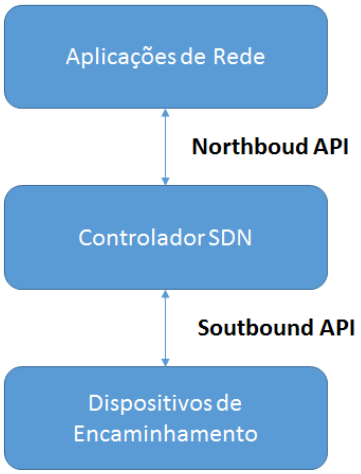

Figura 2.2: Diagrama das APIs de uma SDN.

Em relação à *Northbound* API, visto não existir nenhum protocolo de padronização é importante referir que muitos controladores implementam a sua própria interface. Outro aspeto no qual é diferente da *Southbound* API é que a *Northbound* API é geralmente um ecossistema de software e não de hardware [9]. A *Northbound* API expõe uma visão global da rede, que é mantida pelo controlador, às aplicações para usufruírem da abstração concedida e ser possível realizar software de maneira mais simples.

2.2.3 Protocolo OpenFlow

Existem vários protocolos para a *Southbound* API, como por exemplo o POF [23], ForCES [24], Open vSwitch Database (OVSDB) [25], OpFlex [26] e OpenState [27]. Contudo, o protocolo mais popular e utilizado nas SDN é o OpenFlow [12]. Tanto o OpenFlow como a arquitetura SDN começaram como um projeto académico [28] e têm ganho importância com grande parte dos vendedores de *switches* comerciais a incluir suporte da API do OpenFlow nos seus equipamentos.

A figura 2.3 apresenta a constituição de um *switch* OpenFlow que é constituído por um canal de comunicação, por uma ou mais *flow tables*, uma *group table*, e por uma *meter table*. O OpenFlow é um protocolo orientado a *flows*, como tal é necessário definir o que é uma *flow-entry* para a compreensão correta do que acontece num *switch* OpenFlow.

Uma *flow-entry* é uma regra que dita ao *switch* OpenFlow como atuar quando recebe um pacote de dados. A *flow-entry* é transmitida aos *switches* OpenFlow pelo controlador através do canal de comunicação. As *flow-entries* são armazenadas nas *flow tables*.

Na versão mais recente do OpenFlow [29], as *flow-entries* contêm:

- ***Match Fields*:** Os campos de correspondência servem para identificar quais os pacotes a que se aplicam esta *flow-entry*.

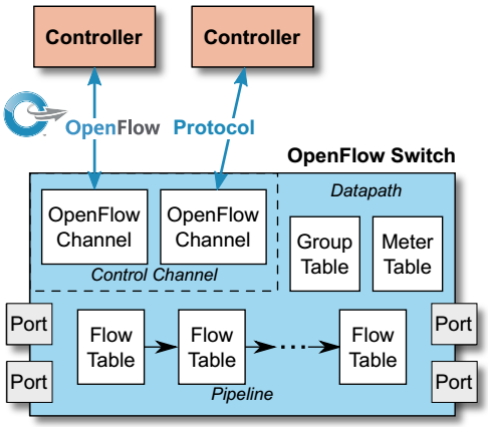

Figura 2.3: Modelo OpenFlow. (Adaptado de [29])

- ***Priority*:** O campo da prioridade serve para definir a ordem pela qual as *flow-entries* estão organizadas nas *flow-tables*.
- ***Counters*:** Estatísticas sobre o tráfego a que a *flow-entry* foi aplicada, tais como número de pacotes, número de bytes, etc.
- ***Instructions*:** O campo das instruções indica qual a ação a realizar ao pacote.
- ***Timeouts*:** Neste campo pode ser introduzido um *hard timer* que indica o tempo de vida da *flow-entry* ou um *idle timer* que indica após quanto tempo de inatividade uma *flow-entry* deve ser removida.
- ***Cookie*:** É um valor opaco escolhido pelo controlador, que pode ser usado pelo mesmo para filtrar as *flow-entries*.
- ***Flags*:** As *flags* alteram a maneira como uma *flow-entry* é gerida.

As ações que um *switch* OpenFlow pode realizar sobre um pacote são bastante simples: encaminhar o pacote para um porto local com a possibilidade de alterar algum campo do cabeçalho do pacote primeiro; ou descartar o pacote; ou por fim, encaminhar o pacote para o controlador.

Uma *group table* é constituída por *group-entries*. Uma *group-entry* consiste num identificador de grupo, um tipo de grupo, contadores, e uma lista de *action buckets*, onde cada *action bucket* contém um conjunto de ações para serem executadas [29]. O que acontece é que as *flow-entries* podem apontar para um grupo oferecendo métodos adicionais de encaminhamento, como por exemplo, *broadcast*.

Uma *meter table* consiste em *meter entries*, que definem *meters* por cada *flow*[2]. Os *meters* por *flow* permitem ao OpenFlow limitar a taxa de débito da rede, mais concretamente, uma simples operação de *Quality of Service* (QoS) que restringe o conjunto de *flows* a

[2] *Flow* é o fluxo de tráfego que circula entre dois nós que partilham as mesmas caracteristicas.

uma largura de banda. Também permitem a implementação de operações de QoS mais complexas, como por exemplo classificar pacotes de acordo com a taxa de débito [29].

A comunicação entre o controlador e o *switch* OpenFlow é realizada através de um canal de comunicação. Este canal pode ser um simples canal de *Transport Control Protocol* (TCP) ou pode ser encriptado com *Transport Layer Security* (TLS) para efetuar comunicações seguras. Um *switch* OpenFlow pode ter mais do que um canal para o controlador, de forma garantir uma maior robustez na rede.

Dentro de cada *switch* OpenFlow, quando um novo pacote chega, o processo de correspondência começa na primeira tabela e existem duas possibilidades: ou é encontrada uma correspondência ou não. No caso de existir uma correspondência, o pacote é tratado consoante a ação que está na *flow-entry*, mas na situação contrária existem duas ações possíveis: ou o pacote é descartado, ou é enviado para o controlador. O pacote só é enviado para o controlador caso exista uma regra padrão, *default rule*, que indica que todos os pacotes novos devem ser encaminhados para o controlador. No caso da não existência desta regra, todos os pacotes que chegam ao *switch* OpenFlow e que não tenham correspondência são descartados.

É importante referir que um *switch* OpenFlow não é necessariamente um dispositivo de hardware, sendo possível a sua implementação com software. Na realidade *switches* implementados através de software estão a crescer como uma solução promissora para *data centers* e infraestruturas de rede virtualizadas [9]. Exemplos de software que permitem a implementação de *switches* OpenFlow incluem o Switch Light [30], ofsoftswitch13 [31], Open vSwitch [32] e Pica8 [33]. Estudos recentes mostram que o número de portos de acesso virtuais já excedeu os portos de acesso físico em *data centers* [34].

A figura 2.4 representa a arquitetura SDN com OpenFlow:

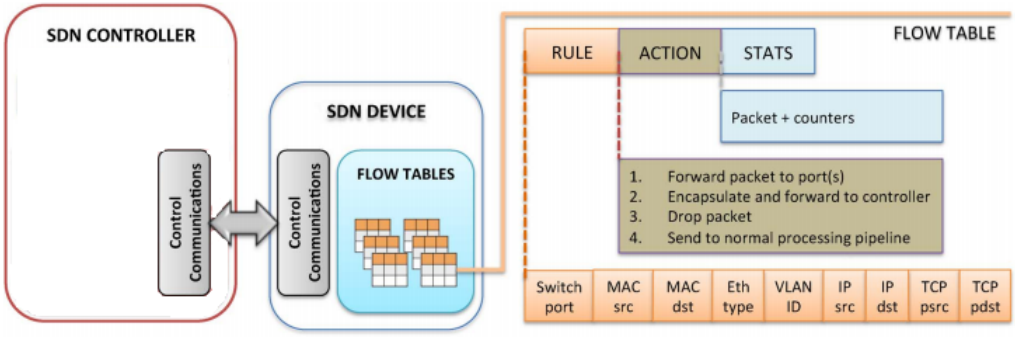

Figura 2.4: Arquitetura SDN com OpenFlow. (Adaptado de [9])

Antes da implementação prática de uma SDN com OpenFlow, existem algumas ferramentas de simulação para provar o conceito, como por exemplo NS-3 [35] conjugado com OpenFlow Software Implementation Distribution (OF-SID) [36]. Outra ferramenta de simulação é o Mininet [37] que utiliza o Open vSwitch [32] para construir redes SDN totalmente em software para teste.

2.2.4 Otimização em SDN

Usando SDNs, os problemas de otimização tradicionais apresentados em 2.1.2 tornam-se menos complexos de resolver. Têm surgido alguns trabalhos como [38] e [4] que visam resolver os problemas de otimização MCF de forma a aumentar a eficiência da rede. Contudo as soluções apresentadas necessitam da matriz de tráfego para servir como *input*, algo que nem sem sempre é possível, e padecem de problemas de escalabilidade [1].

Um dos problemas é a complexidade da implementação de uma solução ótima, caso seja possível, e o número de regras que resultam da otimização [1]. Tal como em MPLS-TE, o uso de *explicit routing* simplifica a solução. No entanto, em MPLS-TE a solução ótima traduzia-se na implementação de múltiplos *túneis*, algo que em SDN se traduz em regras a inserir nos *switches* OpenFlow. O número de regras que se pode implementar num *switch* OpenFlow é limitado, como tal, é um recurso crítico a ter em consideração na implementação de uma solução de encaminhamento.

Outro problema de escalabilidade é o *control-loop*, ou seja a comunicação entre o controlador e os *switches* OpenFlow, pois numa solução que se adapte aos requisitos de tráfego o algoritmo de PL tem de ser corrido cada vez que os requisitos mudam [1]. Esta comunicação entre o controlador e os *switches* OpenFlow pode deteriorar o desempenho da rede introduzindo atraso, sendo que a isto acresce a dificuldade de atualizar as regras em todos os *switches* OpenFlow.

Tanto a Google como a Microsoft implementaram WANs definidas por software que interligam os seus *data centers*, onde em ambos os casos os resultados da utilização de SDN foram positivos trazendo grandes benefícios ao desempenho da rede [38], [4]. Foi formulado o problema de otimização de MCF nos dois projetos, contudo a solução ótima de encaminhamento resultou num grande número de regras a serem implementadas nos *switches* OpenFlow, mesmo para redes de um tamanho consideravelmente pequeno. Perante esta situação, foi necessário realizar-se uma adaptação ao problema de encaminhamento formulado utilizando uma solução sub-ótima.

Perante os problemas de escalabilidade introduzidos pela implementação de uma solução de encaminhamento ótima, e perante a complexidade dos algoritmos de PL, torna-se pertinente estudar outras metodologias partindo da necessidade de manter um número reduzido de regras e uma comunicação entre o controlador e os *switch* OpenFlow pouco frequente de modo a obter uma solução escalável.

Desta forma, e tirando partido do nível de programação da rede do paradigma SDN, podem explorar-se outras formas de encaminhamento alternativas ao tradicional caminho mais curto, que mantenham o encaminhamento baseado apenas no destino mas que aumentem as possibilidades de gestão de tráfego.

2.3 Descrição da Arquitetura SDN

SDN introduz novos níveis de programação e flexibilidade da rede, simplificando o problema da eficiência do transporte de dados nas redes. Contudo é necessário ter em consideração os problemas de escalabilidade no desenho de uma aplicação SDN, de forma a conseguir tirar partido de todas as vantagens inerentes ao seu uso. Neste cenário é bastante importante definir uma arquitetura que possa contornar estes problemas de forma a obter uma solução de encaminhamento escalável e robusta.

Nesta secção apresenta-se uma arquitetura de controlo hierárquica de dois níveis em conjunto com regras de encaminhamento baseadas no destino como solução para se obter um conjunto reduzido de *flow-entries* e reduzir a comunicação entre o plano de controlo e os SDN-FE.

2.3.1 Plano de Controlo

O conhecimento global da rede e o uso de caminhos explícitos por *flow* simplifica o problema de otimização MCF, contudo o número de *flow-entries* necessárias para a implementação de uma solução ótima e a frequência da comunicação entre o controlador e os SDN-FE (*control-loop*) tornam a solução pouco escalável. Assim sendo é necessário reformular o problema de encaminhamento e utilizar soluções sub-ótimas [38], [4]. Os problemas referidos são dois dos principais problemas de escalabilidade, no entanto existem outras preocupações a ter em conta no planeamento de uma solução baseada em SDN, nomeadamente o *overhead* na implementação de *flow-entries* e a resistência da rede a falhas, tal como nas redes tradicionais [39].

Uma arquitetura possível para o plano de controlo é colocar toda a lógica de controlo num controlador central da rede. Desta forma consegue-se uma visão completa da rede tornando-se mais simples desenvolver aplicações de controlo e aplicar políticas de rede [39]. Contudo, o uso de uma arquitetura completamente centralizada pode tornar o controlador num ponto de estrangulamento da rede, sendo que à medida que a rede cresce, o número de eventos e pedidos direcionados ao controlador aumenta de tal forma que o controlador deixa de conseguir satisfazer todos os pedidos. Soluções como o DevoFlow [40] usam uma abordagem de controlo centralizada, onde apenas as *flows grandes* são encaminhadas para o controlador. Neste cenário, existe uma troca entre a visibilidade no controlador, tendo apenas visão sob *flows grandes*, e escalabilidade.

Uma arquitetura mais interessante é a distribuição da lógica de controlo por múltiplos controladores, removendo o ponto de estrangulamento da rede e aumentando a escalabilidade. Soluções como o HyperFlow [41] fornecem uma ilusão de controlo sob toda a rede através da sincronização do estado da rede entre os vários controladores. Desta forma mantém-se a simplicidade no desenvolvimento de aplicações para o plano de controlo aliviando a carga no mesmo [39]. A robustez do sistema é inerente ao uso de uma arquitetura de controlo distribuída, onde em caso de falha de um controlador um SDN-FE

consegue sempre descobrir outro controlador.

A implementação de *flow-entries* nos SDN-FE tem um custo associado (*overhead*). A sua implementação de forma puramente reativa garante à rede um grande nível de flexibilidade, no entanto introduz um grande atraso, limitando a escalabilidade do sistema [39]. Implementar *flow-entries* de forma pró ativa reduz o atraso introduzido pela implementação de *flow-entries* de forma puramente reativa, contudo reduz a flexibilidade do sistema.

Nesta dissertação foi usada uma arquitetura de controlo hierárquica de dois níveis, figura 2.5, que foi inicialmente introduzida em [1]. Existem dois tipos de controladores, o controlador responsável por gerir os nós de acesso (ANC) e o controlador responsável pelos nós do *core* (CNC). Como estes tipos de controlador são logicamente separados um do outro, a sua implementação pode ser realizada em máquinas diferentes e em diferentes pontos na rede [1].

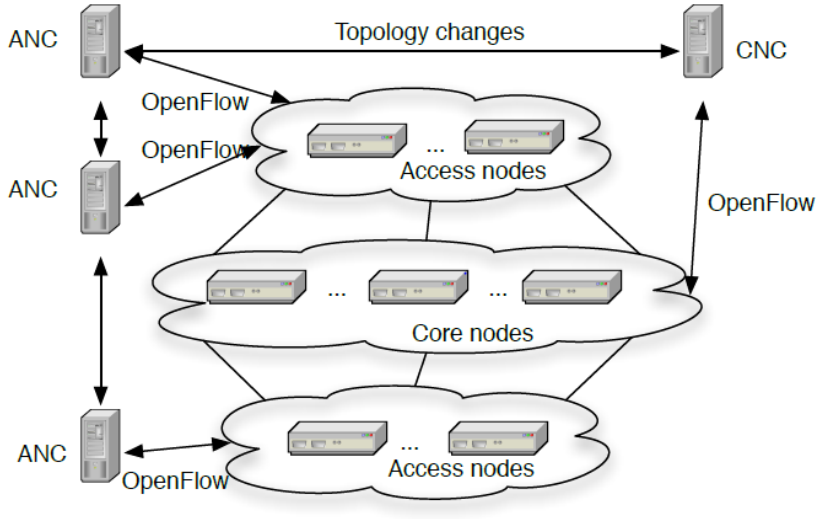

Figura 2.5: Arquitetura hierárquica do plano de controlo. (Adaptado de [1])

O motivo da separação é simples: como os nós do *core* estão sujeitos a uma maior quantidade de tráfego que os nós de acesso, se utilizarmos uma implementação de *flow-entries* reativa para satisfazer os diferentes pedidos irá ser instalado um grande número de *flow-entires* e será necessário existir muita comunicação entre o controlador e os SDN-FE. O que é precisamente o que se pretende evitar, pois o espaço para implementar *flow-entries* é um recurso limitado e crítico nos SDN-FE, e muita comunicação entre o controlador e os SDN-FE introduz *delay* na rede, podendo colocar o controlador como um ponto de estrangulamento da rede. Neste cenário, nos nós do *core* são introduzidas regras pró-ativamente de forma a minimizar a comunicação entre o CNC e os SDN-FE, sendo que só existe comunicação caso ocorram falhas na rede. Os pacotes que não correspondam a nenhuma *flow-entry* serão descartados e nenhuma mensagem OpenFlow é enviada para o CNC, visto que não existe nenhuma função reativa no CNC [1].

Por outro lado, os nós de acesso estão sujeitos a uma quantidade de tráfego muito

menor que os nós do *core* portanto estão menos restringidos que os nós do *core* em relação ao número de *flow-entries* que se pode implementar. Desta forma é possível existir maior comunicação entre o controlador e os SDN-FE, visto que os nós de acesso apenas lidam com tráfego que geram ou que lhes é direcionado. Como consequente é utilizada uma implementação de *flow-entries* de forma reativa. Visto que estes nós estão sujeitos a uma menor quantidade de tráfego, a comunicação com o controlador também será menor que nos nós do *core*. Como tal os nós de acesso são responsáveis por fornecer informação das *flow-entires* implementadas. Assim sendo, torna-se possível inferir qual a utilização dos *links* da rede.

O uso de uma arquitetura hierárquica de dois níveis, onde é utilizado uma implementação pró-ativa e reativa de *flow-entries* permite obter o nível de flexibilidade desejado e reduzir o *delay* na implementação de *flow-entries*.

O ANC necessita de ter comunicação com os SDN-FE usando o protocolo OpenFlow e se for usado mais que um ANC, é também necessário existir comunicação com os restantes ANCs usando uma API própria [1]. No caso do CNC, controla os nós do *core* usando OpenFlow. Tanto o ANC como o CNC têm conhecimento da topologia da rede, através de configuração ou mensagens OpenFlow próprias onde é necessário existir comunicação entre os dois tipos de controlador [1].

2.3.2 Definição das *flow-entries* de encaminhamento

Tanto o ANC como o CNC executam o algoritmo responsável pelo cáculo dos caminhos igualmente preferidos entre os nós de acesso da rede. Após o cálculo dos caminhos, os mesmos são utilizados pelos controladores para definirem as *flow-entries* nos SDN-FE. As *flow-entries* são instaladas nos SDN-FE com base no endereço destino, sendo que quando existe mais que um caminho entre um par de nós de acesso é instalada uma regra por caminho [1].

De forma a conseguir distinguir os diferentes caminhos entre um par de nós de acesso, as *flow-entries*, na sua essência, são constituídas por dois campos de correspondência: o endereço destino e uma *tag*. As *tags* são implementadas com base nos caminhos igualmente preferidos, sendo que uma *tag* representa o caminho completo por onde um pacote irá circular. Neste cenário, o número de *flow-entries* instaladas num SDN-FE é igual ao número de caminhos igualmente preferidos entre o conjunto de nós de acesso da rede, resultando no mesmo número de regras que em encaminhamento tradicional.

Nos nós de acesso as *flow-entries* são implementadas de forma reativa, sendo que são os nós de acesso responsáveis por implementar uma *tag* nos pacotes. Quando um pacote sem *tag* chega a um nó de acesso, o nó envia uma mensagem OpenFlow que contém o pacote para o ANC. O ANC escolhe qual o caminho a utilizar e define uma *flow-entry* onde os campos de correspondência são o endereço IP de destino e endereço IP de origem, e as ações são colocar a *tag* correspondente ao caminho e reencaminhar o pacote para o porto correspondente.

De forma a exemplificar o descrito apresenta-se a figura 2.6, onde estão representados um conjunto de nós de acesso que interligam um conjunto de *hosts*. O ANC e o CNC calculam os caminhos entre os nós de acesso da rede e vão a cada nó da rede implementar as *flow-entries* necessárias para o uso dos caminhos. No caso dos *hosts* a roxo existem dois caminhos possíveis de se utilizar, o vermelho e o azul, e cada um desses caminhos tem uma *tag* que os distingue. Se o nó de acesso coloca a *tag* 100, que corresponde ao caminho vermelho, num pacote então o pacote irá seguir o caminho vermelho. Caso o nó de acesso coloque a *tag* 101 num pacote, então o pacote irá seguir o caminho azul. Desta forma é possível distribuir o tráfego por ambos os caminhos.

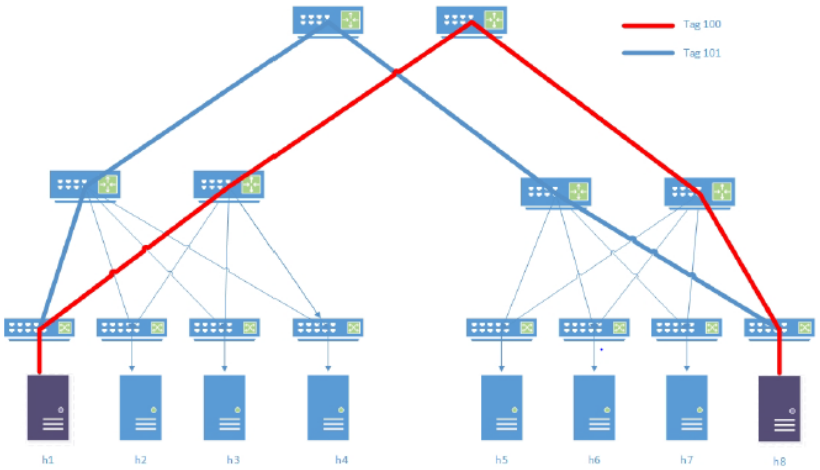

Figura 2.6: Exemplo do uso de *tags*. (Adaptado de [1])

2.4 Modelos de Encaminhamento Algébrico

A utilização de modelos algébricos permite o desenho de novos algoritmos de encaminhamento com elegância. Com esta secção pretende-se explicar como é que estes modelos surgem assim como a sua importância. Alguns conceitos de álgebra linear são introduzidos para uma melhor compreensão dos termos utilizados em modelos de encaminhamento algébrico usados ao longo do relatório.

A álgebra não clássica é bastante utilizada em vários ramos informáticos, tais como programação, criptografia, desenho de compiladores, conseguindo assimunir conceitos que à primeira vista não estão relacionados e estabelecer resultados fundamentais [42]. Com a utilização de estruturas algébricas adequadas e a exploração das respetivas propriedades, a utilização de uma abordagem algébrica mostra ter grande impacto na expressividade de políticas de rede em protocolos de encaminhamento.

As políticas de rede refletem um conjunto de características com conceitos semanticamente mais ricos que métricas numéricas, definindo a natureza dos caminhos e a sua

preferência relativa. Como consequência, caminhos topologicamente diferentes podem ser considerados igualmente "bons", aumentando potencialmente o número de caminhos utilizáveis entre os nós. Com uma abordagem algébrica, a tarefa de definir um protocolo de encaminhamento e provar o seu funcionamento torna-se bastante clara.

2.4.1 Bases Algébricas

Os conceitos de álgebra linear têm sido usados em vários campos científicos, em particular na Física e Matemática Aplicada, utilizando estruturas algébricas clássicas nomeadamente *groups*, *rings* e *fields*. No entanto, existem situações onde a utilização de estruturas algébricas clássicas pode não fornecer as ferramentas necessárias para modelar e apresentar soluções em áreas especificas [43]. Assim sendo, surge a necessidade de se utilizarem estruturas algébricas mais primitivas que se apresentam como mais apropriadas para a resolução de certos problemas em áreas específicas. Um exemplo típico é a estrutura algébrica modelada pelo conjunto dos inteiros positivos dotada da adição normal e multiplicação, que não contém todas as propriedades de um *field* nem de um *ring*, contudo é uma estrutura de interesse [44].

Estruturas como a exemplificada ou até estruturas mais primitivas podem ser utilizadas para modelar problemas de encaminhamento. A razão principal é que as operações de compor caminhos e posteriormente decidir a prioridade atribuída a cada caminho não têm propriedades tão completas como as impostas em *groups*, *fields* ou *rings* [43]. As estruturas mais primitivas são formadas por um conjunto dotado de duas operações binárias para modelar as operações de encaminhamento, nomeadamente $\otimes$ e $\oplus$.

Tendo este conceito em mente, iremos considerar estruturas algébricas da seguinte forma genérica:

$$(S, \oplus, \otimes)$$

Esta estrutura generalizada representa uma estrutura mais primitiva quando comparada com *fields*, ou até *rings*. Neste cenário, a operação $\otimes$ modela a operação de cálculo dos caminhos e a operação $\oplus$ modela a seleção dos caminhos. De forma a efetuar a seleção dos caminhos a operação $\oplus$ tem de ser idempotente [44], ou seja, $a \oplus a \forall a \in S$. No caso particular onde a operação $\oplus$ introduz uma ordem no conjunto, podemos ter um estrutura na forma ordenada:

$$(S, \preceq, \otimes)$$

Onde a ordem $\preceq$ é dada por $\oplus$ como: $a \preceq b$ se $a \oplus b = a$, sendo este o caso típico em modelos de encaminhamento onde a operação $\oplus$ é a operação *Max* ou *Min* [44].

Para descrever o comportamento de protocolos de encaminhamento, estruturas primitivas como as descritas anteriormente têm de ser utilizadas. Compreender sob que condições um protocolo de encaminhamento opera corretamente é fundamental para a projeção de novas soluções de encaminhamento.

2.4.2 Modelo Teórico

De uma forma geral diz-se que um protocolo de encaminhamento tem um comportamento correcto se conseguir calcular um conjunto de caminhos em tempo finito. Esta definição mais geral não tem em consideração o encaminhamento realizado pelo protocolo, se existe a ocorrência de *loops* ou não. Com base em [45] iremos definir o funcionamento correto de um protocolo tendo os dois aspetos em mente, sendo que consideramos que um protocolo está correto se cumprir com a definição adotada em [45].

Definição 2.1 *Um protocolo de encaminhamento é correto se e só se:*

- Se converge para uma solução de encaminhamento, significando que calcula um conjunto fixo de caminhos em tempo finito;
- Se realiza o encaminhamento dos pacotes através de qualquer um dos caminhos igualmente preferidos sem causar qualquer tipo de *loop* (no nosso caso, usando encaminhamento *hop-by-hop* baseado no destino).

Para conseguir compreender sob que condições um protocolo de encaminhamento consegue operar corretamente é necessário a utilização de um modelo de encaminhamento de forma a provar as condições de convergência e o correto funcionamento de uma forma geral.

O problema de encaminhamento é geralmente modelado de uma forma puramente algorítmica, misturando a componente que modela o comportamento desejado para o protocolo e a componente algorítmica de encaminhamento [44]. Quando se aborda um problema de encaminhamento onde se pretende atribuir pesos para os caminhos mais abstratos, ou seja, não numérico, recorre-se à teoria de grafos com auxilio a álgebra linear para se modelar o problema. Neste caso, a componente que define o comportamento desejado para o protocolo é modelada algebricamente,utilizando-se teoria de grafos para modelar a rede.

Com a utilização de teoria de grafos podemos representar a rede como um grafo direcionado $G(V, E)$, onde V representa um conjunto de vértices, os nós constituintes da rede, e E um conjunto de arestas, ou seja, as ligações entre os nós da rede. Sendo necessário modelar [45]:

- O conjunto dos atributos das ligações;
- O processo da obtenção de caminhos através da conjugação de ligações;
- O processo de seleção dos caminhos a serem utilizados.

Tendo a abstração da rede em mente e os aspetos necessários a modelar, é necessário definir uma estrutura algébrica para modelar as políticas de rede a implementar. Uma estrutura que é usada regularmente, sendo que também foi a primeira estrutura algébrica utilizada para modelar protocolos de encaminhamento é o *semiring* [43].

As propriedades dos *semirings* são adequadas para modelar o encaminhamento tradicional do caminho mais curto, sendo conhecido que um protocolo de encaminhamento modelado por um *semiring* funciona de forma correta e produz um conjunto de caminhos globalmente ótimos [43], [46], [47], [48]. No entanto, o encaminhamento baseado em políticas de rede torna a tarefa de modelação mais complicada, sendo que se perdem algumas propriedades dos *semirings* [45].

Assim sendo, uma estrutura algébrica mais geral, ou seja, menos restritiva, formada por um conjunto S dotado de duas operações binárias $\oplus$ e $\otimes$ associativas, conhecida como *bisemigroup* é mais adequada para a modelação de um protocolo de encaminhamento baseado em políticas de rede [44]. Consgue-se assim reter a simplicidade de encaminhamento *hop-by-hop* baseado no destino e podendo realizar o encaminhamento dos pacotes por qualquer um dos caminhos mais preferidos [45].

Definição 2.2 *Um bisemigroup é uma estrutura algébrica constituída por um conjunto S dotado de duas operações binárias associativas [44], $\oplus$ e $\otimes$, sendo que temos associatividade de $\oplus$*

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c \forall a, b, c \in S$$

e associatividade de $\otimes$

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c \forall a, b, c \in S$$

O conjunto S modela os atributos, ou seja, o valor atribuído às ligações, e o valor dos caminhos utilizados para o encaminhamento dos pacotes [44]. $\oplus$ tem um elemento identidade, $\bar{0}$, que modela uma caminho não existente ou inválido. $\bar{0}$ é um aniquilador de $\otimes$, ou seja, quando se adiciona uma ligação a um caminho inválido ou não existente o caminho resultante continuará inválida ou não existente, $a \otimes \bar{0} = \bar{0}$ [42], [44]. A operação $\otimes$ também possui um elemento identidade, $\bar{1}$. Este elemento modela um caminho trivial, que é o caminho de um nó para ele mesmo, não contendo ligações com outros nós [42], [44].

Com este modelo é possível modelar a operação de seleção de caminhos colocando uma ordem de preferência em todos os valores dos caminhos possíveis em S. Considerando o seguinte exemplo de $a \preceq b$, o valor a é preferido ao valor b se $a \oplus b = a$, definindo assim a ordem canónica $\preceq$.

Temos que $\bar{1} \preceq a \preceq \bar{0}$, ou seja o caminho trivial é sempre o caminho mais preferido, e intuitivamente o caminho não existente ou inválido é o menos preferido. Os valores atribuídos aos caminhos obtidos pela operação $\otimes$ encontram-se sempre entre estes dois casos [44]. Se substituirmos a operação $\oplus$ por $\preceq$ temos uma estrutura conhecida como *ordered semigroup* [47]:

$$(S, \preceq, \otimes)$$

Escolher entre *bisemigroups* ou *ordered semigroups* é uma questão de se considerar a seleção do caminho como o resultado de uma operação binária sobre o custo de dois caminhos ou o resultado de uma ordem definida à priori pelo gestor de rede de modo a definir a preferência dos caminhos obtidos, modelando assim o comportamento da rede.

2.4.3 O Problema de Encaminhamento

De acordo com a definição 2.1, o primeiro aspeto a verificar se o protocolo tem um comportamento correto é confirmar se o protocolo consegue encontrar os melhores caminhos disponíveis de um nó para todos os destinos da rede em tempo finito. Este problema é o problema de encaminhamento e caso exista uma solução então a primeira condição imposta pela definição 2.1 é confirmada.

A rede é modelada por um grafo direcionado $G(V, E)$, onde V representa um conjunto de vértices, os nós constituintes da rede, e E um conjunto de arestas, ou seja, as ligações entre os nós da rede. É necessário considerar uma função que atribui os pesos às ligações, ou seja, uma função peso. Iremos denotar esta função peso como $w(i, j)$, que faz a projeção de $E \rightarrow S$, ou seja, atribui valores às ligações entre os nós. A rede é representada por uma matriz de adjacência A [48], onde $A[i, j] = w(i, j)$, ou seja:

$$A[i,j] = \begin{cases} w(i,j), & (i,j) \in E \\ 0, & \text{caso contrário} \end{cases}$$

Considere-se um caminho uma sequência de vértices, $P = v_1, v_2, \ldots, v_k$, sendo que o seu custo é calculado através da combinação dos pesos de todas as ligações entre os nós, combinação essa que é dada pela função $\otimes$ da seguinte forma, $w(P) = w(v_1, v_2) \otimes w(v_2, v_3) \otimes \ldots \otimes w(v_{k-1}, v_k)$. Se considerarmos $P(i, j)$ o conjunto de todos os caminhos entre os nós i e j, então, o problema de encaminhamento consiste em encontrar o conjunto $O(i, j)$ dos caminhos de maior preferência em $P(i, j)$ de acordo com a operação $\oplus$, para todos os pares $(i, j) \in V$ [44]. Sendo o conjunto $O(i, j)$ interpretado da seguinte maneira:

$$O(i,j) = \bigoplus_{p \in P(i,j)} w(p)$$

É possível encontrar a solução para este problema resolvendo um de dois problemas [44]. O problema de encaminhamento *esquerdo* e o problema de encaminhamento *direito*. A diferença entre as soluções locais do problema de encaminhamento *esquerdo* e *direito* é a ordem do cálculo do peso do caminho: começando do destino para origem no caso *esquerdo*, ou da origem para o destino no caso *direito*.

O problema de encaminhamento *esquerdo* é dado por [44]:

$$L(i,j) = \bigoplus_{q \in V} A(i,q) \otimes L(q,j)$$

que corresponde a resolver a seguinte equação matricial [43]:

$$L(i,j) = (A \otimes L) \oplus I \tag{2.1}$$

$L(i, j)$ indica o conjunto dos pesos dos caminhos de maior preferência entre os nós i e j tendo em consideração os pesos dos caminhos de maior preferência entre qualquer nó q da origem i.

O problema de encaminhamento *direito* é dado por [44]:

$$R(i,j) = \bigoplus_{q \in V} R(i,q) \otimes A(q,j)$$

que corresponde a resolver a seguinte equação matricial [43]:

$$R(i,j) = (R \otimes A) \oplus I \tag{2.2}$$

De forma análoga à equação anterior, $R(i,j)$ é o conjunto dos pesos dos caminhos de maior preferência tendo em consideração os pesos dos caminhos de maior preferência entre qualquer nó q do destino j.

A solução para ambos os casos é encontrada calculando a matriz A^*, que indica o peso ótimo do caminho entre dois nós, isto é, $A^*[i,j]$ indica o peso ótimo entre os nós i e j. O cálculo da matriz A^* é realizado sob certas condições de convergência expressas por algumas das propriedades algébricas das estruturas *ordered semigroup* [43]. A^* é obtida através da iteração sucessiva de matrizes, aplicando $\otimes$ a A e minimizando o resultado obtido de cada elemento da matriz segundo a ordem de preferência dada por $\preceq$ [43], [46]. Apesar de A^* apenas indicar quais os pesos ótimos dos caminhos entre os nós, durante o seu cálculo é possível armazenar através de que nós se obtém o peso pretendido e desta forma definir o caminho entre os nós. A existência A^* garante que o protocolo de encaminhamento converge para um solução de encaminhamento estável [44]. O cálculo da matriz A^* é geralmente associado aos algoritmos de Bellman-Ford e Dijkstra que são conhecidos para conseguirem calcular a solução para os problemas de encaminhamento referido anteriormente que correspondem a resolver as equações (2.1) e (2.2) [43], [48].

2.4.4 Condições de Convergência

De forma a tornar possível o cálculo de A^* é necessário considerar um conjunto de condições. A primeira condição garante que o protocolo converge em qualquer rede e é conhecida como monotonia. A monotonia implica que o custo de um caminho não decresce quando se adiciona outra ligação ao caminho [42].

Definição 2.3 *Um ordered semigroup* $(S, \preceq, \otimes)$ *é monótono se [44]:*

$$a \preceq b \Rightarrow (c \otimes a) \preceq (c \otimes b) \forall a, b, c \in S$$

Um bisemigroup $(S, \oplus, \otimes)$ *é monótono se [44]:*

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \forall a, b, c \in S$$

Se considerarmos o caso onde o encaminhamento é modelado utilizando um *ordered semigroup*, de acordo com a ordem imposta por $\preceq$, o caminho mais preferido é o caminho que tem o melhor peso. O melhor peso representa o mínimo considerado em $\preceq$, como tal um sub-caminho do melhor caminho é também ele um melhor caminho [44]. A monotonia

é uma condição suficiente para garantir que o protocolo converge, garantindo sempre a existência de A^*.

As restantes condições estão relacionadas com a operação $\otimes$, onde existem duas possibilidades. Uma operação $\otimes$ com propriedade *crescente* ou $\otimes$ com propriedade *não decrescente*. No caso de considerarmos uma operação $\otimes$ *crescente*, é uma condição mais restritiva que a monotonia e não implica a monotonia, contudo é uma condição suficiente para garantir a existência de A^*.

Definição 2.4 *Uma operação $\otimes$ é crescente se [45]:*

$$a \prec a \otimes b \forall a, b \in S$$

A utilização de uma operação $\otimes$ *crescente* implica que a preferência de um caminho decresce sempre quando se adiciona uma ligação. No caso de apenas se considerar uma operação $\otimes$ *crescente* e não se considerar a monotonia, o cálculo de A^* ainda é possível. Neste cenário a solução das equações (2.1) e (2.2) podem não ser iguais [44] e correspondem a um ótimo local [42], [43], [46] onde os melhores caminhos num nó dependem dos melhores caminhos de outros nós, não garantindo que os sub-caminhos dos melhores caminhos sejam também melhores caminhos.

Definição 2.5 *Uma operação $\otimes$ é não decrescente se [45]:*

$$a \preceq a \otimes b \forall a, b \in S$$

A utilização de uma operação $\otimes$ *não decrescente* por si só não garante a convergência do protocolo, sendo necessário garantir restrições na atribuição dos elementos de S á rede [45].

Apesar de não se garantir convergência com uso de uma operação $\otimes$ *não decrescente*, por si só, é de interesse a capacidade de se adicionar ligações a um caminho e manter a sua preferência. Neste cenário, abre-se a possibilidade da existência de caminhos com diferente numero de ligações, mas com a mesma preferência, potencialmente aumentado o número de caminhos utilizáveis entre dois nós. Sendo uma estratégia a explorar na maximização da utilização dos recursos da rede.

O impacto da rigidez da monotonia na convergência para uma solução de encaminhamento

A definição 2.3 garante a existência de A^* e que o protocolo converge, contudo não assegura a convergência em tempo finito. Sendo necessário analisar o impacto da rigidez da monotonia na convergência para uma solução de encaminhamento.

Definição 2.6 *Um ordered semigroup é estritamente monótono se [44]:*

$$a \prec b \Rightarrow (c \otimes a) \prec (c \otimes b) \forall a, b, c \in S$$

De acordo com a definição 2.3, a preferência de um caminho pode ser mantida com a adição de ligações, como consequência podem existir ciclos[3] na rede onde um número infinito de caminhos têm todos o melhor custo. Apesar de existir convergência para uma solução global ótima, não existe convergência em tempo finito [44]. Para a garantir convergência em tempo finito a monotonia tem de ser rígida, como apresentado na definição 2.6.

2.4.4.1 Condições de Convergência em Encaminhamento Multi-Caminho

De forma a maximizar a utilização dos recursos da rede, a utilização de encaminhamento multi-caminho é um aspeto chave. Com encaminhamento multi-caminho o encaminhamento dos pacotes é realizado por mais do que um caminho em simultâneo para o destino, tornando-se importante derivar as condições de convergência para o caso de multi-caminho. É importante verificar se são semelhantes às condições de convergência quando se encaminha os pacotes por apenas um caminho.

Existem dois problemas a ter em consideração: o problema do *k-melhor caminho* e o problema do *multi-caminho* [44]. No problema do *k-melhor caminho* existe mais do que um caminho entre (i, j) que têm o mesmo custo, com $w \in S$. O problema do *multi-caminho* refere-se à existência de mais do que um caminho com custos igualmente preferidos, ou seja, existe mais do que um peso que têm a mesma preferência na ordem $\preceq$, com $w_1, w_2 \in S$.

Apesar das diferenças, ambos os problemas podem ser modelados definindo conjuntos mínimos do conjunto S [44]. Um conjunto mínimo representa um conjunto de caminhos de igual preferência, e iremos representar o conjunto de todos os conjuntos mínimos do conjunto S como $M(S)$. Ao considerarmos caminhos com a mesma preferência estamos a implicar que ou os caminhos têm o mesmo custo $w_1 = ... = w_n$ ou têm um custo equivalente $w_1 \simeq ... \simeq w_n$.

De acordo com [44] é introduzida a seguinte definição:

Definição 2.7 *Um conjunto mínimo é um subconjunto $C \subset S$ com cardinalidade $|C| = n$ com:*

$$C = min_{\preceq}(C) = \{x \in C | \forall y \in C : x \preceq y\}$$

É possível definir outra estrutura algébrica derivada de um *ordered semigroup* onde se usa os conjuntos mínimos e os operadores binários são redefinidos de forma a trabalhar com os conjuntos mínimos [44]. Considerando-se $\oplus'$ na seguinte forma:

$$A \oplus' B = min_{\preceq}(A \cup B) \tag{2.3}$$

O que nos indica é que o operador $\oplus'$ devolve os caminhos mais preferidos da união de A com B. Da mesma forma é necessário redefinir $\otimes$, sendo que a operação que calcula os caminhos pertencentes aos conjuntos mínimos, $\otimes'$, é dada por:

$$a \otimes' C = min_{\preceq}(a \otimes B) \forall a \in S, C \in M(S) \tag{2.4}$$

[3]Considera-se um ciclo um caminho onde o primeiro é também o último nó.

Ou seja, a operação $\otimes'$ resulta nos caminhos de menor custo quando se adiciona uma nova ligação, a, a todos os elementos do conjunto mínimo C para todos os conjuntos mínimos existentes. Com as operações binárias redefinidas chega-se à estrutura algébrica necessária para modelar multi-caminho:

$$(M(S), \oplus', \otimes')$$

No trabalho realizado em [49] é provado que se uma álgebra for monótona o conjunto mínimo de encaminhamento pode ser encontrado e é um ótimo global. Em adição, prova que na ausência de monotonia, uma álgebra apenas com uma operação $\otimes$ crescente é uma condição suficiente para garantir uma solução ótima local utilizando encaminhamento multi-caminho, independentemente do problema de encaminhamento considerado. Concluindo-se que as condições de convergência são iguais para encaminhamento multi-caminho ou encaminhamento *single path*.

Apesar de uma álgebra monótona garantir convergência para um ótimo global, é difícil manter a monotonia em protocolos baseados em políticas de rede [44]. Neste cenário, introduz-se uma operação $\otimes$ estritamente crescente que introduz uma limitação no uso de possíveis caminhos na rede, diminuindo assim a diversidade de caminhos a utilizar. Como apresenta uma limitação no aumento da eficiência da utilização dos recursos da rede, reforça o interesse numa solução baseada numa operação $\otimes$ *não decrescente*. Contudo, com o uso de uma operação $\otimes$ *não decrescente* podem existir ciclos na rede que sejam considerados igualmente preferidos a outros caminhos, sendo necessário garantir que não existem ciclos problemáticos na rede.

Estes ciclos podem ser formalizados como uma generalização da condição de ciclos *não negativos* [44]. A primeira generalização formal deste tipo de ciclos é conhecida como uma *roda de disputa* e foi introduzido por [50]. O trabalho realizado em [42] introduz uma generalização mais algébrica que é referenciada como *ciclo livre*. Em [44] é usada outra definição do *ciclo livre* que irá ser adotada neste trabalho, desta maneira definimos um *ciclo livre* em álgebra pura para um *ordered semigroup*.

Definição 2.8 *Considere-se um ciclo* $P = v_1, v_2, \ldots, v_(k-1), v_k = v_1$. *Considere-se d um destino,* A_i *o conjunto de todos os caminhos do vértice* v_i *para d, e* $a_{ij} \in S - \bar{0}$ *o peso do caminho j que se inicia no vértice i para o destino d e que passa pelo nó* $v_(i+1)$. *Desta forma,* $a_{ij} \in A_i$. *Considere-se o conjunto de todos os caminhos para d a partir de todos os vértices do ciclo,* $A_1 \cup A_2 \cup \ldots \cup A_{k-1} \cup (A_k = A_1)$. *O ciclo é considerado livre se existir pelo menos um caminho j com inicio no nó i* $(1 \leq i \leq k-1)$ *onde* $a_{ij} \prec w(v_i, v_{i+1} \otimes a_{(i+1)k}) \forall k \in A_{i+1}$. *Esta condição tem de ser válida para todos os destinos alcançáveis, d.*

A figura 2.7 ilustra o conceito de *ciclo livre* e de acordo com a definição 2.8, para um ciclo ser considerado livre pelo menos um dos caminhos a tracejado tem de existir e tem de ser mais preferido no nó v_i do que o caminho com as linhas sólidas para v_{i+1} adicionado através da operação $\otimes$ para o próximo caminho a_{i+1j} , ou outro [44]. Esta

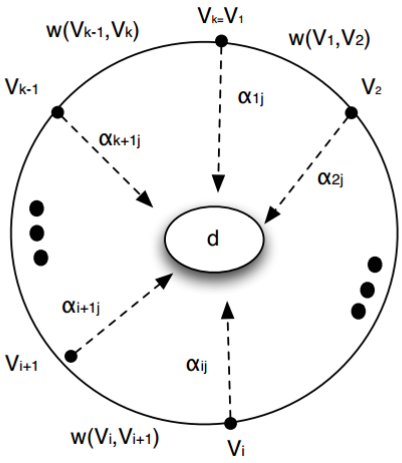

Figura 2.7: Ciclo livre. (Adaptado de [44])

condição implica que existe pelo menos um caminho fora do ciclo que é mais preferido aos caminhos dentro do ciclo.

A ausência de ciclos *não livres* pode ser obtida por propriedades algébricas ou através de restrições no grafo da rede. No caso de se utilizar uma operação $\otimes$ *crescente* não existem ciclos *não livres* por definição, sendo que no caso de se utilizar uma operação $\otimes$ *não decrescente* é necessário realizar restrições no grafo da rede. A convergência para uma solução de encaminhamento em tempo finito pode ser assegurada para um protocolo modelado por um *ordered semigroup* com uma operação $\otimes$ *não decrescente* se a rede tiver as restrições necessárias para garantir que todos os ciclos são *livres*. A prova deste resultado pode ser encontrada em [44].

De forma a garantir a convergência é necessário introduzir um método que consiga identificar se existem, ou não, ciclos *não livres* na rede. Esse método foi definido por [44] e é definido por:

Definição 2.9 *Para cada valor $s \in S - \bar{0}$ considere-se um conjunto $L_s \subset S, L_s = \{l \in S | s = l \otimes s\}$. Isto significa que um conjunto L_s contém os valores possíveis de se atribuírem às ligações que mantêm a preferência quando se adiciona uma ligação. Um ciclo é livre numa álgebra com uma operação* $\otimes$ não decrescente *se para cada L_s, o ciclo tem pelo menos uma das suas ligações que não pertence a esse conjunto. Ou, um ciclo é não livre caso todas as suas ligações pertençam a um dos conjunto L_s.*

2.4.4.2 Condições de Convergência em Encaminhamento hop-by-hop baseado no destino

Até agora foram apresentadas as condições necessárias para garantir a convergência para uma solução de encaminhamento em tempo finito. O correspondente ao protocolo encontrar os melhores caminhos de acordo com a ordem $\preceq$, contudo, um aspeto a considerar

após a obtenção dos melhores caminhos é o encaminhamento dos pacotes. Pretende-se neste trabalho manter a simplicidade de encaminhamento *hop-by-hop* baseado no destino, como tal é importante verificar sob que condições se consegue um encaminhamento sem *loops*, que são causados por decisões de encaminhamento inconsistentes.

Em álgebras estritamente monótonas como apresentado na definição 2.6 não existe este problema, pois os sub-caminhos são também eles caminhos ótimos, contudo se se descartar a monotonia, *loops* podem acontecer. Nesta situação é necessário distinguir as equações (2.1) e (2.2), onde na equação (2.1) a operação $\otimes$ é aplicada para a *esquerda* e a equação (2.2) onde a operação $\otimes$ é aplicada para a *direita*. Na ausência de monotonia a solução das equações é diferente, e no caso de se resolver o problema de encaminhamento *direito*, a sua solução pode causar *loops* em encaminhamento *hop-by-hop* baseado no destino [48]. A razão para tal é que na equação (2.2) o custo dos caminhos é calculado da origem para o destino e como o próximo nó pode ser diferente, um *loop* pode ocorrer. No caso da equação (2.1) o cálculo do custo é feito do destino para a origem e mesmo na ausência de monotonia, visto que o custo dos caminhos dependem do custo de sub-caminhos dos próximos nós para o destino, não ocorrem *loops* [44].

Capítulo 3

Descrição do Projeto

O uso de uma arquitetura SDN adequada permite desenvolver uma aplicação SDN escalável e robusta. Usando a arquitetura SDN apresentada no capítulo anterior e tirando partido das vantagens do uso de SDN surgem novas oportunidades de estudo, nomeadamente: como calcular os caminhos entre os nós de acesso e como distribuir o tráfego por esses mesmos caminhos.

Nesta dissertação foram desenvolvidos vários *scripts* em Python como modo de estudo dessas alternativas e uma aplicação em Java com o objetivo de realizar simulações próximas da realidade e validar os resultados obtidos. Neste capítulo é descrita a implementação em Python e Java realizadas, começando pela implementação em Python.

3.1 Implementação da ferramenta de modelação teórica

De forma a conseguir estudar vários tipos de encaminhamento de forma simples e eficaz implementou-se uma ferramenta de modelação teórica. Recorreu-se à linguagem de *scripting* Python e à biblioteca igraph [51] para o desenvolvimento da mesma. A biblioteca igraph é uma coleção de ferramentas de análise de redes *open source* e grátis que foi utilizada nesta dissertação para o estudo de grafos.

3.1.1 Criação dos grafos

O uso de um grafo que represente as redes que se pretendem utilizar é o primeiro passo para se conseguir estudar novos métodos de encaminhamento. A biblioteca de estudo de grafos usada, o igraph, facilita imenso este passo. Através da matriz de adjacência representativa de uma rede é possível utilizar o igraph para criar um grafo $G(V, E)$, onde V são os nós da rede e E as ligações entre os nós. É utilizado um método implementado pela biblioteca igraph que lê a matriz de adjacência de um ficheiro de texto, onde após

a leitura da matriz de adjacência são criadas automaticamente as estruturas de dados representativas dos nós e das ligações entre eles.

Por cada nó e por cada ligação é criada uma estrutura do tipo dicionário. As estruturas do tipo dicionário conferem um nível alto de flexibilidade na configuração de objetos, sendo bastante simples adicionar atributos aos nós e às ligações, como por exemplo configurar a largura de banda das ligações.

3.1.2 Cálculo dos caminhos entre os nós de acesso

Para além da forma mais comum de encaminhamento, que consiste em resolver o problema do caminho mais curto onde o custo de um caminho é uma métrica numérica, foi estudada um alternativa que consiste no uso de uma abordagem de encaminhamento baseada em políticas de rede. Políticas de rede definem um conjunto de características com conceitos semanticamente mais ricos que métricas numéricas.

Para se conseguir exprimir políticas rede é necessário recorrer ao uso de modelos algébricos. Uma estrutura algébrica é representada da seguinte forma:

$$(S, \oplus, \otimes)$$

O conjunto S modela os atributos da rede, nomeadamente: o valor atribuído às ligações e o custo dos caminhos entre os nós de acesso. A operação $\oplus$ modela a seleção dos caminhos e a operação $\otimes$ modela o cálculo dos caminhos. No caso específico onde o resultado da operação $\oplus$ é uma ordem pré-definida, as estruturas algébricas tomam o seguinte aspeto:

$$(S, \preceq, \otimes)$$

De acordo com a definição 2.1, um protocolo de encaminhamento é *correto* se conseguir convergir para uma solução de encaminhamento e se conseguir realizar o encaminhamento de pacotes sem causar qualquer tipo de *loop*. Contudo, a arquitetura SDN utilizada permite que a álgebra utilizada seja mais simples. O uso de *tags* que identificam um caminho completo entre os nós de acesso confere ao protocolo a realização do encaminhamento de pacotes sem *loops*.

Utilizou-se o protocolo de encaminhamento baseado em políticas hierárquicas introduzido em [44] como método de estudo da variação no cálculo de caminhos entre os nós de acesso. Como se pretende realizar encaminhamento multi-caminho, o protocolo deve ser modelado recorrendo a uso de um conjunto mínimo sob um *ordered semigroup* [44].

O conjunto S do protocolo usado é constituído por seis elementos:

- $\bar{1}$ - caminho trivial;
- $\bar{0}$ - caminho inválido;
- D_w - caminhos e ligações onde o nó de origem tem um nível hierárquico maior que o destino;

$\otimes$	$\bar{1}$	D_w	S_{LR}	S_{LL}	U_w
$\bar{1}$	$\bar{1}$	D_w	S_{LR}	S_{LL}	U_w
D_w	D_w	D_w	$\bar{0}$	$\bar{0}$	$\bar{0}$
S_{LR}	S_{LR}	S_{LR}	S_{LR}	$\bar{0}$	$\bar{0}$
S_{LL}	S_{LL}	S_{LL}	S_{LL}	S_{LL}	$\bar{0}$
U_w	U_w	U_w	U_w	U_w	U_w

Tabela 3.1: Operação $\otimes$

- U_w - caminhos e ligações onde o nó de origem tem um nível hierárquico menor que o destino;
- S_{LL} - caminhos e ligações entre nós no mesmo nível hierárquico mas numa direção específica;
- S_{LR} - caminhos e ligações entre nós no mesmo nível hierárquico mas numa direção oposta a S_{LL}.

Inicialmente o conjunto S continha apenas o valor S_L para os nós que se encontram no mesmo nível hierárquico [44]. Contudo, o uso de um único atributo para nós no mesmo nível hierárquico poderia causar *loops* de encaminhamento caso a atribuição das *labels* não fosse realizada de forma correta [45]. Por consequência o atributo S_L foi substituído por: S_{LL} e S_{LR}. Apesar de não ser necessário a álgebra adotada verifica ambas as condições especificadas na definição 2.1.

O conjunto de pares de *labels* a aplicar às ligações entre os nós são os seguintes:

- $[D_w - U_w]$ - para ligações entre nós com um nível hierárquico diferente;
- $[S_{LR} - S_{LL}]$ - para ligações entre nós no mesmo nível hierárquico.

A operação $\oplus$ é dada pela seguinte ordem de preferência:

$$\bar{1} \prec D_w \prec S_{LR} \prec S_{SLL} \prec U_w \prec \bar{0}$$

O que significa que os caminhos com o valor D_w são os mais preferidos e os caminhos com o valor U_w são os menos preferidos, ou seja as soluções de encaminhamento local são preferidas a soluções que passem pelo *core* da rede [45]. Por fim, a operação $\otimes$ é definida na tabela 3.1 onde existe um conjunto de casos que resultam num caminho inválidos. Com a invalidação de caminhos pretende-se reduzir a quantidade de conjunto de valores que pertencem a S, $L_{[u]}$, específicos a cada classe de equivalência, que quando são sujeitos à operação $\otimes$ mantêm a preferência da classe. Isto é:

$$L_{[u]} = l \in S | com(l \otimes u) \in [u]$$

Na álgebra adotada estes valores são:

- $L_{[D_w]} = D_w$

- $L_{[S_{LR}]} = S_{LR}$
- $L_{[S_{LL}]} = S_{LL}$
- $L_{[U_w]} = U_w$

O trabalho realizado em [45] prova que esta álgebra confere todas as condições necessárias para convergir para uma solução de encaminhamento em tempo finito e para realizar encaminhamento multi-caminho sem causar qualquer tipo de *loop*.

A forma como os elementos do conjunto S são atribuídos às ligações entre os nós da rede mostra ter impacto na quantidade de tráfego transmitido na rede, sendo que a sua atribuição também foi alvo de estudo. Foram usadas métricas de grafos tais como *eigenvector centrality* e *node degree* para realizar a atribuição dos elementos de S às ligações da rede.

A métrica *node degree* indica a quantidade de ligações que um nó tem. Numa situação em que se considera a rede como uma hierarquia, nós que possuam um *degree* alto podem ser considerados como estando num nível alto da hierarquia. A métrica *eigenvector centrality* mede a influência de um nó na rede, um exemplo de uma variante desta métrica é o sistema de *ranking* das páginas da Google. No nosso caso, um nó com uma *eigenvector centrality* alta indicia que é um nó influente na rede e por consequência num nível alto da hierarquia. Ao atribuir níveis aos nós, quando se percorre as ligações existentes na rede é possível atribuir um elemento do conjunto S olhando para os nós que uma dada ligação interliga.

Algoritmo de exploração de grafos

Após a definição dos métodos de cálculo de caminhos a utilizar, caminho mais curto e políticas de rede, foi necessário implementar um algoritmo de exploração de grafos. Com este algoritmo pretende-se saber através de que nós se consegue chegar a um destino e desta forma conseguir o construir caminho até esse destino. O algoritmo implementado nesta fase é conhecido como *Breadth-First Search* (BFS) [52]. O algoritmo tem como parâmetros de entrada o grafo representativo da rede e um nó de acesso, e tem como parâmetro de saída os destinos alcançáveis por esse nó. O nó de acesso de que se pretende saber os destinos é conhecido como *root*. No nosso caso, os destinos alcançáveis pelos nós de acesso serão também eles nós de acesso. Este algoritmo calcula os destinos alcançáveis tendo em consideração qual o método de cálculo de caminho a utilizar. Por exemplo, no caso do algoritmo que resolve o problema do caminho mais curto os nós destinos já serão alcançáveis por outros nós que constituirão o caminho mais curto para o destino. Desta forma pretende-se obter um novo grafo da rede, mais pequeno que o grafo que representa a topologia da rede, que represente os caminhos desejados.

O primeiro passo após a escolha do nó *root* é considerar a distância para todos os outros nós como infinito. De seguida verifica-se quem são os nós vizinhos do nó *root*,

nesta fase calcula-se o custo de comunicação entre o nó *root* e os vizinhos. Como o custo de comunicação entre o nó *root* e os seus vizinhos é menor que infinito considera-se o nó *root* como um *pai* dos seus vizinhos. O termo *pai* é usado para descrever um nó a qual dá acesso a outro nó, neste caso *filho*. Os nós vizinhos são adicionados a uma *queue* para depois se poderem percorrer. Após se visitar todos os nós vizinhos do nó *root* é necessário verificar quem são os vizinhos dos nós foram colocados na *queue*. Seleciona-se o primeiro nó, o qual iremos designar de *current*, que se encontra na *queue* e vê-se quem são os vizinhos, excluindo o nó *pai* de forma a evitar *loops*. Nesta fase o processo é idêntico ao descrito anteriormente, isto é, calcula-se o custo de comunicação entre o nó *root* e os novos nós. Caso algum dos nós considerados já tenha sido alcançado existem três hipóteses de progressão:

- Se o custo atual é menor que o custo anterior, então considera-se o nó *current* como *pai* do nó e atualiza-se o custo a que se chega a este nó;
- Se o custo atual é maior que o custo anterior, então descarta-se esta possibilidade;
- Se o custo atual é igual ao custo anterior, então atualiza-se a lista de nós *pais* adicionando-se o nó *current*.

Os nós vizinhos do nó *current* são adicionados à *queue*. Este processo repete-se até se conseguir encontrar todos os nós alcançáveis pelo nó *root* e de se considerar todos os nós de acesso como *root*.

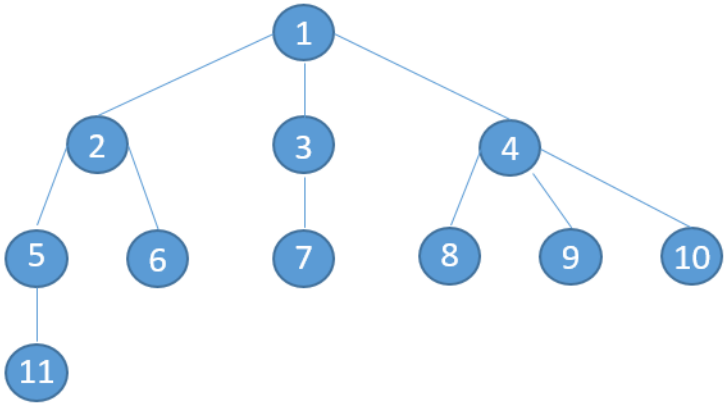

Figura 3.1: Exemplo da execução do algoritmo BFS.

A figura 3.1 ilustra um exemplo da execução do algoritmo para uma melhor compreensão do mesmo. O nó 1 é o nó *root* e é o primeiro nó a ser considerado. Os seus vizinhos são os próximos nós a considerar pela ordem apresentada na figura. Calcula-se o custo de comunicação entre o nó 1 e os vizinhos e considera-se o nó 1 *pai* dos mesmos. Os nós vizinhos são adicionados a uma *queue* e o processo repete-se. O nó 11 é o último nó a ser visitado.

Construção dos caminhos

O uso do algoritmo BFS permite saber quais os destinos alcançáveis por um nó *root*. A informação armazenada consiste nos nós que permitem alcançar outros nós e o custo de comunicação entre o nó *root* e os destinos. Sempre que se conclui a pesquisa de destinos de um certo nó é preciso construir os caminhos até aos destinos. Neste caso a construção é realizada no sentido inverso à exploração do grafo, isto é, começa-se no destino e vai-se percorrendo o conjunto de *pais* dos nós até se chegar ao nó *root*.

A ideia é simples, começa-se no destino pretendido e vê-se o conjunto de *pais* do destino. Concatena-se os *pais* ao destino e obtém-se um conjunto de caminhos incompletos. Depois vai-se aos *pais* do destino e obtém-se os seus *pais* e concatena-se aos caminhos incompletos. Concatenando-se os nós de forma recursiva obtém-se os caminhos até à origem. Este processo só é possível visto que o algoritmo de exploração de grafos calcula os *pais* de forma a evitar *loops*.

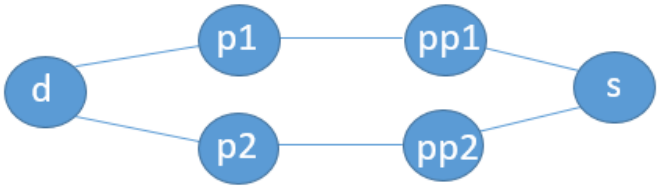

Figura 3.2: Exemplo da execução do algoritmo de construção dos caminhos.

Na figura 3.2 temos um nó destino, o nó *d*, que tem dois *pais*, os nós *p1* e *p2*. O algoritmo de construção de caminhos inicialmente constrói dois caminhos incompletos, que são constituídos pelo o nó *d* e os seus *pais*. De seguida, complementa-se a construção dos caminhos adicionado os *pais* dos nós *p1* e *p2*. Finalmente os caminhos são completos pois o nó de origem, o nó *s*, é o *pai* dos nós *pp1* e *pp2*.

Este algoritmo é aplicado sobre um grafo que é construido após se escolher o algoritmo que se irá usar para obter os caminhos mais curtos. Como tal os caminhos obtidos já serão os melhores caminhos.

3.1.3 Cálculo da distribuição de tráfego por caminho

Em técnicas de encaminhamento tradicionais como ECMP o tráfego é dividido de forma equitativa pelos os diferentes caminhos mais curtos existentes entre os pares de nós. No desenvolvimento desta dissertação foi estudada uma alternativa a esta distribuição de tráfego que faz uso de métricas de grafos para influenciar a distribuição do tráfego pela rede de modo a trazer benefícios ao desempenho geral da rede.

Uma métrica interessante e que pode ser usada como uma medida de risco de congestão na rede é a *betweenness centrality*. Indicadores de centralidade identificam a importância dos vértices num grafo, nesta dissertação foi usada a *betweenness centrality* das

ligações que indica a importância de uma ligação na rede.

Considere-se um grafo direcionado $G(V, E)$, onde V representa um conjunto de vértices, os nós constituintes da rede, e E um conjunto de arestas, ou seja, as ligações entre os nós da rede. Adotando a nomenclatura usada em [44], o número de caminhos mais curtos ou igualmente preferidos entre os nós s e t é dado por $\sigma(s, t)$. O número de caminhos mais curtos ou igualmente preferidos entre os nós s e t que têm na sua constituição a ligação e é dado por $\sigma(s, t \mid e)$. Em [53] é apresentada uma fórmula referente à *betweenness centrality* de uma ligação entre dois nós que tem em consideração os caminhos igualmente preferidos entre todos os pares de nós da rede, contudo no nosso caso apenas existe interesse em considerar os caminhos igualmente preferidos entre os nós de acesso. Neste cenário, considere-se $A \in V$ o conjunto de todos os nós de acesso de uma rede. No trabalho realizado em [1] é usada uma alternativa a fórmula apresentada em [53], que indica o valor de *betweenness centrality* considerando apenas os nós de acesso:

$$C_{BA}(e) = \sum_{(s,t)\in A} \frac{\sigma(s, t|e)}{\sigma(s, t)} \tag{3.1}$$

A equação 3.1 é um indicador de risco de congestão de uma ligação pois indica o quão importante é uma ligação face ao fluxo de tráfego da rede [44]. A *betweenness centrality* das ligações por si só não é um critério de interesse. Para realizar a distribuição de tráfego pelos caminhos igualmente preferidos é necessário calcular a centralidade dos caminhos. A centralidade de um caminho calcula-se somando todas as *betweenness centralities* das ligações que constituem os caminhos, isto é [1]

$$C_B(P) = \sum_{e_i \in P} C_{BA}(e_i) \tag{3.2}$$

O valor $C_B(P)$ é usado para realizar a distribuição de tráfego, sendo que entre dois nós é dada preferência aos caminhos com menor valor $C_B(P)$ na distribuição de tráfego visto terem menos risco de congestão. Para conseguir que os caminhos com uma centralidade mais baixa obtenham uma quantidade de tráfego maior é necessário que o cálculo da distribuição de tráfego seja realizada recorrendo a uma distribuição de pesos.

A distribuição de pesos é simples. Para um par de nós de acesso considera-se C_t a soma das centralidades de todos os caminhos disponíveis entre o par de nós, onde a centralidade de um caminho i é dada por C_i. Neste cenário a contribuição de um caminho i para a soma das centralidades, C_t é dado por:

$$Pc_i = \frac{C_i}{C_t} \tag{3.3}$$

O objetivo é que a circulação de tráfego nos caminhos entre dois nós seja maior onde o valor da centralidade seja menor, ou seja, o valor usado para distribuição de tráfego é:

$$Pd_i = \frac{1}{Pc_i} \tag{3.4}$$

Se os valores obtidos em 3.4 forem normalizados por $\sum_{\forall i} Pd_i$, o resultado obtido é a percentagem de tráfego que irá circular nesse caminho.

Uma propriedade interessante do algoritmo que calcula a distribuição de tráfego dos caminhos é a que ajusta a distribuição de tráfego consoante os requisitos da rede. Esta característica é uma alternativa aos métodos tradicionais de cálculo da distribuição de tráfego e irá ser alvo de estudo. Caso um caminho esteja perto do limite de largura de banda disponível, esse caminho não entra nas contas para o cálculo da distribuição de tráfego. Neste cenário, um caminho que esteja perto do limite de largura de banda é considerado como indisponível e o algoritmo atribui uma percentagem de distribuição de tráfego a esse caminho de 0% e recalcula a distribuição de tráfego para os restantes caminhos entre o par de nós afetado.

Como estamos a considerar uma simulação teórica, utiliza-se a largura de banda ocupada nas ligações para inferir quando é necessário bloquear um caminho. No caso de uma aplicação SDN real, de forma a tornar a aplicação escalável apenas são consideradas *flow-entries* grandes para realizar uma estimativa da largura de banda ocupada nas ligações. Neste cenário, o limite considerado para bloquear um caminho é menor do que o limite teórico.

3.2 Implementação de uma aplicação SDN

O desenvolvimento de uma ferramenta de modelação teórica permitiu estudar vários métodos de encaminhamento e realizar simulações teóricas para obter uma estimativa do desempenho dos mesmos. Contudo é necessário validar os resultados obtidos através da realização de simulações em ambientes reais ou em ambientes perto da realidade. O objetivo de implementar uma aplicação SDN em Java é exatamente a validação dos resultados. Como todos os algoritmos necessários já tinham sido implementados em Python, apenas foi necessário transcrever os algoritmos em Java e complementar a aplicação de forma a realizar a implementação de *flow-entries*. Como apenas se pretende validar os resultados obtidos, o algoritmo de cálculo de caminhos não foi implementado em Java. Os caminhos foram lidos de ficheiros de texto criados a partir dos caminhos calculados em Python.

A aplicação desenvolvida é constituída por vários elementos, nomeadamente:

- Um gestor da aplicação;
- Um gestor de caminhos;
- Um conjunto de *listeners* que respondem a eventos da rede;
- Um gestor de pacotes recebidos.

Todos estes elementos são fundamentais para o desenvolvimento da aplicação SDN, sendo que cada um deles tem um conjunto de funções que são descritas nesta secção.

Recorreu-se à versão 1.0 do protocolo OpenFlow e utilizou-se o controlador HP VAN SDN Controller versão 2.0 para desenvolver a aplicação.

3.2.1 Gestor da aplicação SDN

O gestor da aplicação é a classe principal do projeto. Quando se realiza a implementação da aplicação no controlador, a classe do gestor da aplicação é a classe que é chamada. É nesta classe que são iniciados todos os objetos e serviços necessários para a execução da aplicação. Existem dois serviços que são referenciados nesta classe: *ControllerService* e *LinkService*. O *ControllerService* é responsável por abstrair o controlador da rede e o *LinkService* é usado para obter as ligações de um *switch* OpenFlow.

Inicialmente começa-se por iniciar o objeto responsável pela leitura dos caminhos e criação das estruturas de dados necessárias para obter informação sobre os mesmos. De seguida iniciam-se os *listeners*, nomeadamente: o *Switch Listener* e o *Packet Listener*. Os *listeners* permitem reagir a eventos da rede e é nesta classe que se encontra a inteligência da aplicação. Quando se desinstala a aplicação do controlador os *listeners* são desativados.

3.2.2 Gestor dos caminhos

Como apenas se pretende validar os resultados obtidos através dos *scripts* em Python, a inteligência do cálculo dos caminhos e construção dos mesmos não foi implementada. Após o cálculo dos caminhos em Python, os mesmos são escritos em ficheiros de texto. No caso de ECMP os ficheiros de texto apenas contêm a constituição dos caminhos. Por outro lado, numa situação em que se usa a centralidade como medida de distribuição de tráfego os ficheiros contêm informação sobre a constituição dos caminhos e a respetiva centralidade. Como se pretende invalidar certos caminhos quando a largura de banda ocupada de uma ligação se aproxima do limite largura de banda, foi necessário implementar o algoritmo que calcula a distribuição de tráfego por caminho.

Após a leitura dos caminhos o algoritmo de distribuição de tráfego é executado e no final é atribuída uma *tag* a cada caminho. As *tags* são usadas para identificar um caminho entre os igualmente preferidos.

A figura 3.3 apresenta o processo de leitura dos caminhos. O gestor de caminhos tem como parâmetros de entrada o ficheiro a ler e se estamos numa situação de ECMP ou não. É a partir destes dois parâmetros que é possível realizar a leitura dos caminhos. Os algoritmos de leitura dos caminhos e de cálculo da distribuição de tráfego são diferentes consoante estamos numa situação de ECMP ou não. Como tal, o primeiro passo do processo de leitura é verificar efetivamente em qual das situações nos encontramos.

Após a identificação do cenário de teste, ECMP ou não, procede-se à leitura dos caminhos. Sempre que se lê um caminho é criado um objeto personalizado para armazenar a informação do mesmo e o objeto é guardado num *HashMap* para posteriormente se conseguir aceder à sua informação. No final da leitura dos caminhos, calcula-se a distribuição de tráfego correspondente a cada caminho e atribui-se a *tag* a cada caminho.

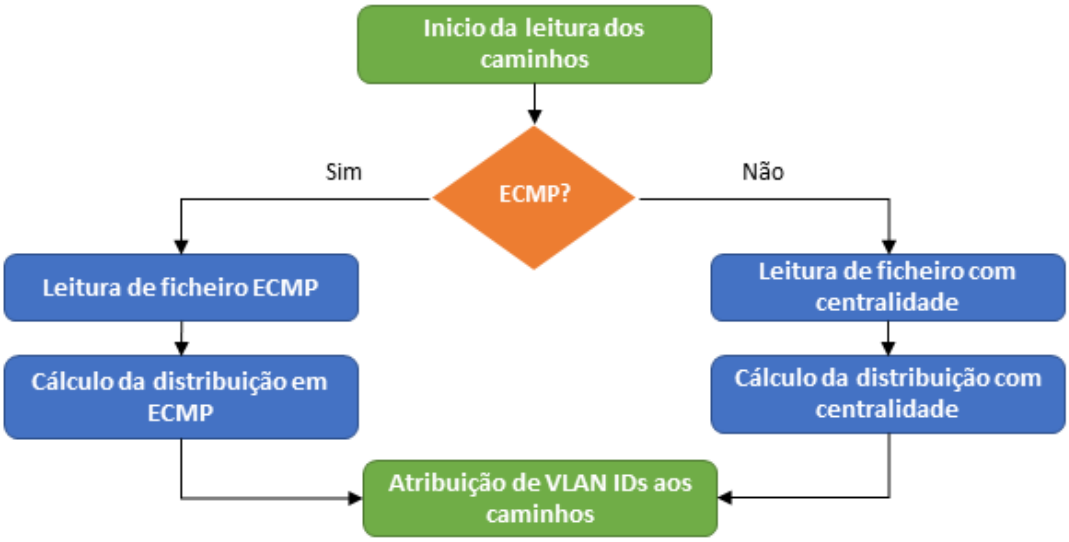

Figura 3.3: Fluxograma da leitura de caminhos.

A leitura dos caminhos é um função importante do gestor de caminhos, contudo existem outras funções que irão ser fundamentais numa fase posterior, como a função de bloqueio e permissão de um caminho que permite que o algoritmo de cálculo da distribuição de tráfego se adapte aos requisitos da rede.

A função de bloqueio de um caminho é chamada quando um caminho tem uma ligação que esteja com uma ocupação de largura de banda perto do limite. O que acontece é que caso existam outros caminhos disponíveis entre a origem e destino do caminho, então o caminho é considerado como inválido e não entra nas contas quando se recalcula a distribuição do tráfego. A função de permissão de um caminho é precisamente o aposto da função anterior. Esta função é utilizada quando um caminho que anteriormente estava indisponível passa a estar disponível.

3.2.3 *Switch Listener*

O *Switch Listener* é uma classe que implementa a interface *DataPathListener* da API da HP. O objeto referente a esta classe é adicionado ao serviço do controlador, o *ControllerService*, como um *DataPathListener* e é através desta classe que consegue responder a eventos relacionados com os *switches* OpenFlow. Os eventos de interesse associados aos *switches* OpenFlow são a sua conexão ao controlador (*DATAPATH_CONNECTED*) e a sua desconexão ao controlador (*DATAPATH_DISCONNECTED*).

O *Switch Listener* tem uma componente pró-ativa e uma componente reativa. O seu funcionamento é explicado no fluxograma da figura 3.4. Inicialmente aguarda-se pelos eventos de conexão dos *switches* OpenFlow ao controlador, até que todos os *switches* OpenFlow da rede se liguem ao controlador. Uma *flow-entry default* é instalada quando um *switch* OpenFlow se liga ao controlador. Esta *flow-entry* indica que quando um pacote

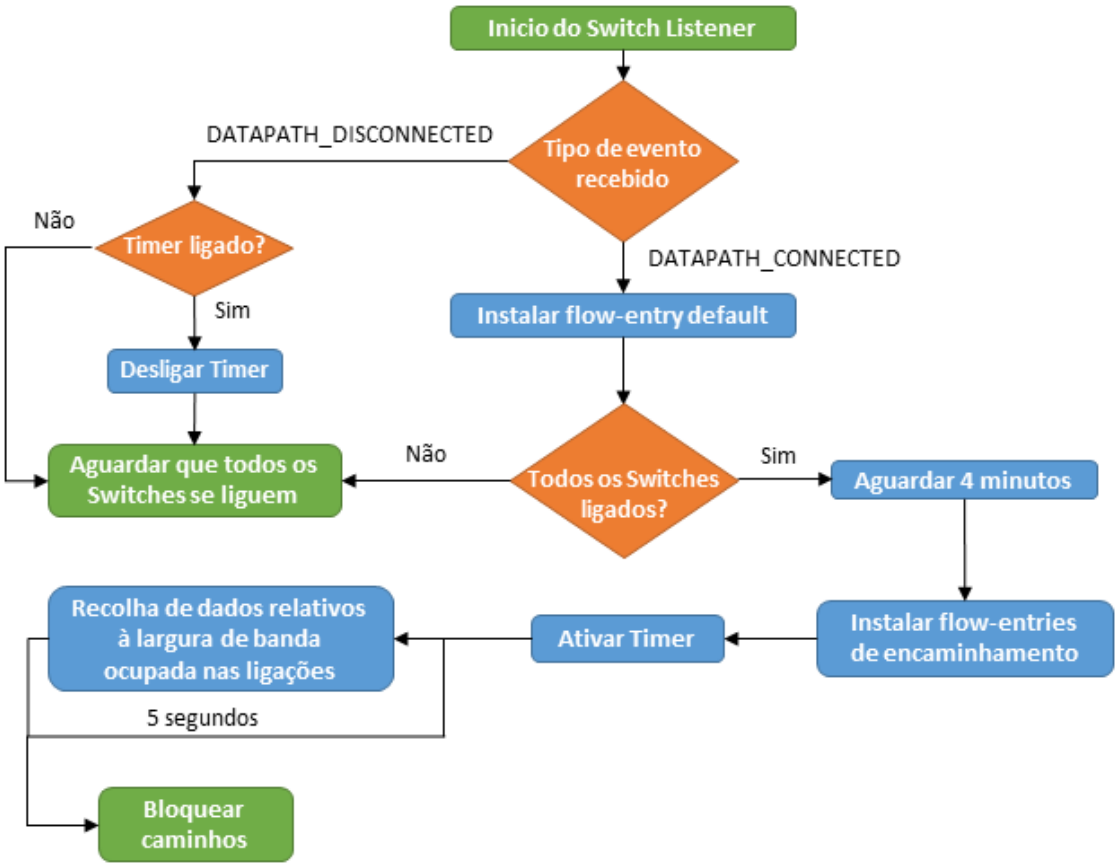

Figura 3.4: Fluxograma do funcionamento do *Switch Listener*.

chega ao *switch* OpenFlow e não existe nenhuma *flow-entry* capaz de satisfazer o pedido, então o pacote deve ser redirecionado para o controlador para se decidir o conjunto de ações a realizar.

Após a conexão de todos os *switches* OpenFlow ao controlador, aguarda-se um período de quatro minutos de forma a permitir a troca de pacotes *Link Layer Discovery Protocol* (LLDP) para exploração da rede, de forma a se conseguir saber as ligações que existem entre os *switches* OpenFlow. No fim do período de espera são instaladas de forma pró-ativa as *flow-entries* necessárias para implementar a solução de encaminhamento a testar.

A componente reativa do *Switch Listener* é garantida por um temporizador que de cinco em cinco segundos recolhe informação relativa à largura de banda que está a ser ocupada nas ligações entre os *switches* OpenFlow. Desta forma é possível inferir quais os caminhos que têm ligações perto de atingir um ponto de congestionamento e se possível considerar esses caminhos como indisponíveis. O temporizador tem um período de espera de cinco segundos pois apenas se pretende inferir a largura de banda ocupada para *flow-entries* grandes. Como consequência, a informação relativa à largura de banda das ligações é apenas uma estimativa.

3.2.3.1 Obtenção da informação relativa à largura de banda ocupada nas ligações

Através do *ControllerService* é possível obter estatísticas das *flow-entries* instaladas nos *switches* OpenFlow que se encontram ligados ao controlador. Pretende-se apenas considerar as *flow-entries* que sejam *grandes*, isto é, que tenham uma duração de pelo menos cinco segundos. Entre os valores possíveis de se obter existe um conjunto que possui interesse para conseguir obter uma estimativa da largura de banda ocupada nas ligações, nomeadamente:

- A lista de ações da *flow-entry*;
- Os campos de correspondência da *flow-entry*;
- A prioridade da *flow-entry*.
- O número de pacotes e o tamanho de todos os pacotes em *bytes* que obtiveram correspondência por parte da *flow-entry*;

A lista de ações da *flow-entry*, os campos de correspondência e a sua prioridade são usadas para conseguir distinguir as *flow-entries* que correspondem à solução de encaminhamento a testar de outras *flow-entries*. Esta distinção é usada para diminuir o tempo de pesquisa das estatísticas das *flow-entries*, pois apenas é necessário obter as estatísticas das *flow-entries* correspondentes à solução de encaminhamento. O número de pacotes é também usado como um filtro para não se perder tempo em obter as estatísticas de *flow-entries* em que nenhum pacote tenha correspondido.

A quantidade de *bytes* de pacotes que corresponderam à regra da *flow-entry* é a estatística que permite inferir a largura de banda ocupada numa ligação. A informação obtida é a quantidade de *bytes* de todos os pacotes que obtiveram correspondência, contudo é usada uma estrutura de dados que armazena a quantidade de *bytes* que tinham obtido correspondência nos cinco segundo anteriores. Desta forma, é possível utilizar um temporizador com um período de cinco segundos, para obter a informação referente à quantidade de *bytes* que obtiveram correspondência e estimar a largura de banda ocupada nas ligações.

Como um caminho completo é identificado por uma *VLAN ID*, de forma a otimizar a obtenção de estatísticas das *flow-entries* infere-se a utilização de uma ligação essa taxa de ocupação é considerada para todas as ligações do caminho correspondente à *flow-entry* considerada.

3.2.4 *Packet Listener*

O *Packet Listener* é uma classe que implementa a interface *SequencedPacketListener* da API da HP. Tal como o *Switch Listener* o objeto referente a esta classe é adicionado ao *ControllerService*, mas como *PacketListener*. Contudo ao contrário do *Switch Listener*, quando se adiciona o *Packet Listener* ao controlador é necessário definir a sua função. Existem três

funções que um *Packet Listener* pode desempenhar: *Adviser*, *Director* e *Observer*. No nosso caso, a função que o *Packet Listener* desempenha é a de *Director*, pois é a função adequada a quem pretende alterar a mensagem *packet-out*. A mensagem *packet-out* é que permite ao controlador enviar as *flow-entries* para os *switches* OpenFlow.

Nesta classe são recebidos os pacotes que não obtêm uma correspondência em nenhuma *flow-entry*, mais precisamente são os pacotes que correspondem à *flow-entry default*. Quando um pacote é recebido é extraída a informação sobre o protocolo do pacote, caso seja o protocolo IP então o pacote é direcionado para a classe *IpPacketHandler* para serem realizadas as ações necessárias.

3.2.4.1 *IpPacketHandler*

A classe *IpPacketHandler* é a classe responsável por inserir *flow-entries* de forma reativa nos *switches* OpenFlow. Após a receção de um pacote é extraído o endereço IP de destino e o endereço IP de origem. Com esta informação é possível consultar a estrutura de dados que possuí as informações sobre os caminhos existentes entre os vários nós de acesso e escolher. Caso só exista um caminho disponível então esse caminho é escolhido, caso existam vários caminhos a sua percentagem de distribuição de tráfego é usada como probabilidade de escolha.

Após a escolha do caminho é construida uma *flow-entry* com os campos de correspondência endereço IP destino e endereço IP origem, sendo que as ações a realizar são colocar uma *VLAN ID* nos pacotes que correspondam à *flow-entry* e encaminhar o pacote para o porto que satisfaça a primeira ligação do caminho. A *flow-entry* possui um *iddle timeout* de cinco segundos.

Capítulo 4

Simulações

A ferramenta de modelação teórica desenvolvida permitiu o estudo de métodos de encaminhamento tais como alterar a forma como são calculados os caminhos e alterar a forma como é realizada a distribuição do tráfego. Em adição ao cálculo dos caminhos e cálculo da distribuição de tráfego, de forma a ter uma referência face ao desempenho dos métodos de encaminhamento estudados, a ferramenta desenvolvida executa uma simulação teórica que produz como *output* um ficheiro de texto com a taxa de transferência de dados média por ligação ativa. De forma a consolidar e verificar os resultados obtidos, foi desenvolvida uma aplicação SDN que realiza as mesmas simulações.

Neste capítulo apresentam-se as simulações realizadas na ferramenta de modelação teórica e na aplicação SDN. As simulações foram realizadas para o pior caso possível de cada método de encaminhamento de cada topologia. Os resultados obtidos nem sempre foram positivos, sendo apresentadas razões para os mesmos. Começa-se por descrever o cálculo do pior caso e de seguida do algoritmo de simulação teórica de forma a se obter uma melhor compreensão dos resultados obtidos.

4.1 Cálculo do pior caso

De forma a se conseguir obter simulações que possam servir como referência face ao desempenho dos métodos de encaminhamento estudados, realizaram-se as simulações considerando o pior caso. Usou-se o algoritmo proposto em [54] para realizar o cálculo do conjunto de pares de nós que quando em comunicação resultam no pior cenário. O primeiro passo para o cálculo do pior caso é calcular um grafo bipartido para cada ligação entre os *switches* OpenFlow da rede. Para o cálculo do grafo bipartido considera-se que todos os nós geradores de tráfego comunicam entre eles e calcula-se o custo que cada comunicação iria ter sobre a ligação. Como não se sabe os requisitos de tráfego, para as

simulações utilizou-se a mesma largura de banda para todas as ligações e considerou-se que todos os nós transmitem à mesma velocidade. Neste cenário o custo de comunicação considerado é a percentagem de tráfego pertencente a cada conexão entre nós que irá circular na ligação.

Após se calcular o grafo bipartido para cada ligação entre os *switches*, o pior caso consiste em resolver um problema de *maximum-weight matching*. Ou seja, calcular qual a ligação que será sujeita a uma maior quantidade de tráfego. O resultado obtido é o conjunto de pares nós que quando em comunicação resultam no valor mínimo a que começa a existir congestão na rede. Para solucionar o problema de *maximum-weight matching* recorreu-se ao algoritmo Munkres [55] que é uma adaptação do algoritmo Hungarian [56]. O pior caso é calculado pela ferramenta de modelação teórica e escrito num ficheiro de texto para depois ser utilizado pela aplicação SDN.

Na figura 4.1 é demonstrada a solução do problema de *maximum-weight matching* para uma ligação l. É apresentada uma tabela que representa o grafo bipartido para uma ligação entre *switches* de uma rede com três nós de acesso. Casa posição da tabela representa o custa que a comunicação entre os nós de acesso considerados tem sobre a ligação. A execução do algoritmo Munkres indica quais os pares de nós que resultam no maior custo para essa ligação. No caso apresentado, o conjunto de comunicações que resultam no pior caso são o nó 0 a transmitir pacotes para o nó 1, o nó 1 a transmitir pacotes para o nó 2, e o nó 2 a transmitir pacotes para nó 0. O custo total que as conexões entre os nós teriam sobre a ligação é de 255%, ou seja, se todos os nós geradores de tráfego se encontrassem a transmitir a por exemplo uma taxa de 100 megabits por segundo, iriam circular nessa ligação 255 megabits por segundo.

	0	1	2
0	-	75%	50%
1	10%	-	80%
2	100%	45%	-

Figura 4.1: Exemplo de um grafo bipartido de uma ligação l.

Como se considera que a taxa de transmissão de pacotes dos nós geradores de tráfego é igual para todos, o valor mínimo a partir do qual começa a existir congestão na rede é dado por:

$$TR_{congestion} = \frac{lb_c}{cl} \times 100\% \tag{4.1}$$

Sendo lb_c a largura de banda da ligação e cl a carga total a que a ligação está exposta. A ligação que obtenha o valor de $TR_{congestion}$ mais baixo representa o pior caso. Existem casos onde o valor mínimo de $TR_{congestion}$ é obtido em diferentes ligações, obtendo-se diferentes pares de nós que representam o pior caso. Neste cenário foram realizadas as simulações para todos os piores casos e no final verificou-se qual dos cenário resultaria

numa taxa de transmissão média menor para várias taxas de transmissão e considerou-se esse o cenário de pior caso.

4.2 Algoritmo de simulação teórica

Com as simulações teóricas pretende-se obter os resultados do desempenho dos métodos de encaminhamento estudados numa situação de convergência, isto é, o tráfego é distribuído consoante os valores calculados para a distribuição de tráfego por caminho. A figura 4.2 ilustra um exemplo da distribuição de tráfego realizada pelo o algoritmo de simulação teórica. Existem três caminhos igualmente preferidos entre o nó *s* que é o nó de origem e o nó *d* que é o nó de destino, cada um com um valor de distribuição de tráfego associado. O nó *s* encontra-se a transmitir a uma taxa de 1000 megabits por segundo e como se trata de uma simulação teórica é possível dividir o tráfego pelos os vários caminhos. O caminho a verde tem uma percentagem de distribuição de tráfego de 70%, como tal estão a circular 700 megabits por segundo. O caminho a amarelo tem uma percentagem de distribuição de tráfego de 20%, logo circulam 200 megabits por segundo e finalmente no caminho a laranja circulam 100 megabits por segundo. Esta distribuição de tráfego é a que se espera numa situação de convergência, onde existem várias conexões ativas entre os pares de nós que constituem o pior caso.

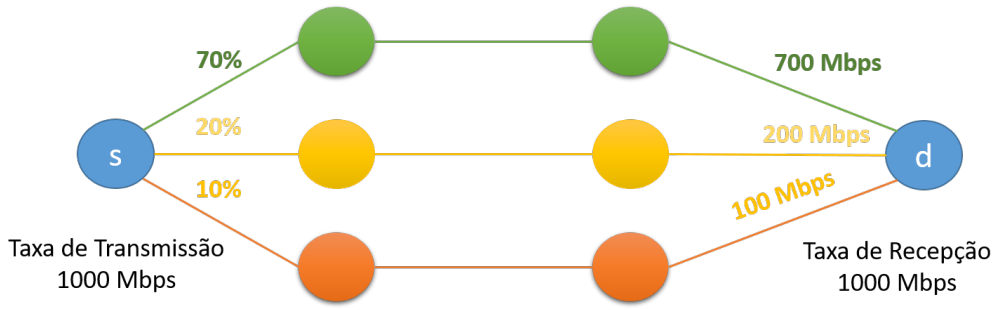

Figura 4.2: Exemplo da distribuição teórica de tráfego.

O algoritmo de distribuição de tráfego por si só não realiza uma simulação completa no caso em que pretendemos testar o bloqueio de caminhos congestionados. Neste cenário o que acontece é que após a simulação de distribuição de tráfego numa ligação é verificada a largura de banda de todas as ligações entre os nós da rede de forma a identificar caminhos congestionados. Quando se deteta uma ligação congestionada a taxa de transmissão a que os nós de acesso estão a transmitir pacotes é reduzida de acordo com a contribuição de pacotes de cada nó de acesso para essa ligação. Desta forma consegue-se punir de forma justa todas as conexões ativas. Este algoritmo repete-se para várias taxas de transmissão dos nós de acesso para se conseguir obter vários resultados.

4.3 Ambiente de simulação SDN

A aplicação SDN tem apenas como objetivo validar os resultados obtidos na ferramenta de modelação teórica, contudo existem algumas diferenças em relação à distribuição do tráfego. No caso da aplicação SDN os caminhos entre dois nós de acesso têm uma percentagem de distribuição de tráfego que deverá circular nesse caminho. Essa percentagem de distribuição é usada como a probabilidade de o caminho ser escolhido para transmissão de tráfego. Após a escolha de um caminho todo o tráfego irá circular nesse caminho até que o tempo de *iddle timeout* da *flow-entry* que coloca a *VLAN ID* nos pacotes seja ultrapassado.

De forma a criar um ambiente de simulação real, usou-se a ferramenta mininet [57]. O mininet é um emulador de redes que permite controlar os parâmetros das ligações entre os nós da rede e que por omissão usa o Open vSwitch para simular os *switches* OpenFlow. Recorreu-se à ferramenta iperf [58], mais propriamente iperf3, para realizar o envio de pacotes entre os *hosts*. É possível criar *scripts* em Python para simular no mininet as redes pretendidas e simular tráfego entre os *hosts* da rede. Realizou-se um *script* em Python que para cada topologia e pior caso para cada método de encaminhamento gera um *script* em Python que implementa comandos da API de Python do mininet.

4.4 Resultados das simulações

O facto de a arquitetura ter sido projetada de forma a ser escalável, aliado ao uso de mecanismos de encaminhamento de baixa complexidade, permite a sua utilização numa grande variedade de redes. O cenário de simulação é igual para todas as redes simuladas. Considera-se que todas as ligações têm uma largura de banda de 1000 megabits por segundo e que todos os nós de acesso transmitem pacotes à mesma velocidade. A taxa de transmissão dos nós de acesso é aumentada gradualmente causando um cenário de congestão e no final da simulação mede-se o valor médio da taxa de receção de pacotes por parte dos nós de destino. As redes utilizadas são baseadas em redes reais ou em redes geradas.

De forma a se conseguir compreender melhor os resultados obtidos é necessário fazer a distinção entre caminhos diferentes e caminhos disjuntos:

- **Caminhos disjuntos:** Caminhos que não partilham qualquer ligação ou nó a não ser o nó de origem e o nó de destino;
- **Caminhos diferentes:** Caminhos que partilham pelo menos uma ligação ou nó, mas contêm pelo menos uma ligação ou nó diferente.

Esta categorização de caminhos é necessária essencialmente para a compreensão dos resultados negativos obtidos. Foram realizadas simulações para três métodos de encaminhamento:

- ECMP - Solução tradicional de ECMP;
- TM - Algoritmo que soluciona o problema do caminho mais curto conjugado com o uso da centralidade para distribuição de tráfego e bloqueio de caminhos congestionados;
- Algebra - Uso de uma álgebra para o cálculo dos caminhos conjugada com o uso da centralidade para distribuição de tráfego e bloqueio de caminhos congestionados. De forma a se conseguir distinguir a forma como a distribuição dos atributos do conjunto S da álgebra usada foi realizada, usa-se o termo Algebra Degree quando os atributos foram distribuídos recorrendo ao *degree* dos nós e Algebra Eigenvector quando a distribuição foi feita com base na *eigenvector centrality*.

A solução tradicional de ECMP foi implementada como método de comparação face às soluções propostas. Os resultados obtidos pela ferramenta de simulação teórica e os resultados obtidos pela aplicação SDN são mostrados lado a lado para se poder comparar os resultados de forma mais simples.

Uma limitação encontrada é o fato de a ferramenta iperf3 conseguir apenas transmitir no máximo cerca de 650 megabits por segundos, as causas para esta limitação são desconhecidas. Face a esta limitação e o facto que nas simulações da aplicação SDN apenas se esteja a simular uma conexão entre os pares de nós que definem o pior caso, tornam previsível que os valores obtidos na aplicação SDN sejam mais baixos que os obtidos através da ferramenta de simulação teórica. Contudo, os resultados obtidos através da aplicação teórica continuam a ser válidos pois servem para confirmar a tendência dos resultados teóricos.

4.4.1 Topologia Abilene

Figura 4.3: Topologia Abilene. (Adaptado de [59])

As primeiras simulações foram realizadas utilizando a topologia da rede Abilene [60], que é uma rede *backbone* de alto desempenho criada pela comunidade Internet2 [61]. Esta topologia é apresentada na figura 4.3 e é constituída por 11 nós em que todos são

considerados nós de acesso. Neste cenário o controlador do tipo CNC coloca as *flow-entries* de forma pró-ativa em todos os nós de acesso, pois para esta rede os nós de acesso são também os nós do *core*. Por consequência perde-se a escalabilidade dada pelo uso de uma arquitetura de controlo hierárquica [1].

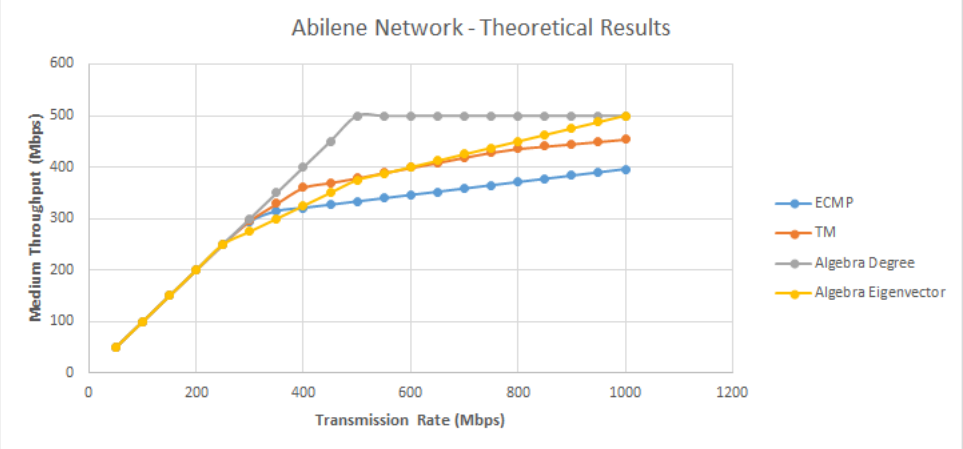

Figura 4.4: Resultados teóricos obtidos para a topologia Abilene.

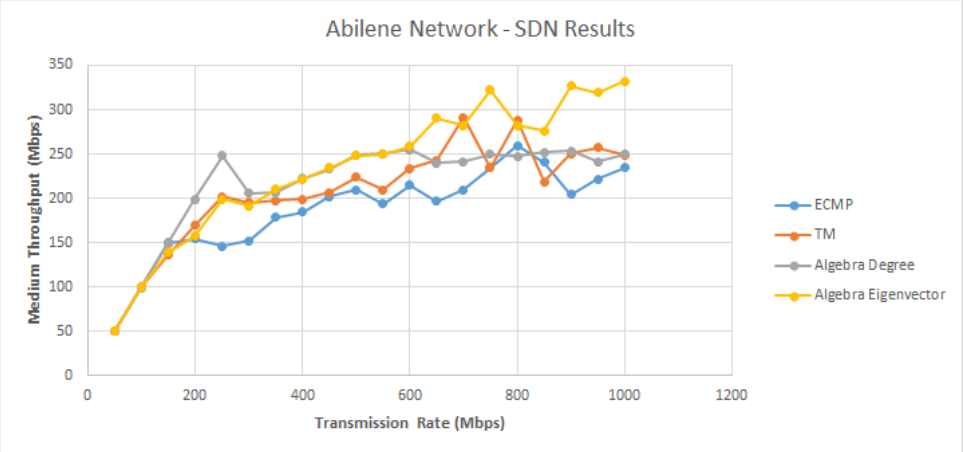

Figura 4.5: Resultados obtidos através da aplicação SDN para a topologia Abilene.

Os resultados das simulações para a rede Abilene encontram-se apresentados na figura 4.4 e figura 4.5. Como esperado, os resultados obtidos através da aplicação SDN têm valores de transmissão média mais baixos que os resultados obtidos através da ferramenta teórica. Contudo é possível verificar os resultados da aplicação SDN seguem a tendência esperada, ou seja vão de acordo com os resultados teóricos.

As abordagem de encaminhamento baseadas em álgebra foram as que obtiveram uma taxa de transmissão média mais elevada, contudo observado a tabela 4.1 é possível verificar que o número de caminhos obtidos nas abordagem baseadas em álgebra são menores que na abordagem baseada no caminho mais curto. Neste cenário, os resultados das abordagens baseadas em álgebra são fruto de que nem todos os nós de acesso conseguem

	Caminho mais curto	Algebra Degree	Algebra Eigenvector
Nº total de caminhos	138	64	114

Tabela 4.1: Número de caminhos entre os nós de acesso obtidos na rede Abilene.

comunicar entre eles por, consequência do anulamento de caminhos utilizado na álgebra, resultando numa quantidade de tráfego a circular na rede menor do que nas abordagens baseadas no caminho mais curto. Como tal, os resultados obtidos para as abordagens algébricas são considerados não válidos.

Comparando o método de encaminhamento TM com a abordagem tradicional de ECMP é possível verificar que nos resultados obtidos através da aplicação SDN, salvo uma exceção, existe claramente um melhor desempenho por parte da abordagem TM. A nível teórico, quando os nós de acesso se encontram a transmitir a 1000 megabits por segundo espera-se que a método de encaminhamento TM consiga transportar cerca de 15% mais tráfego que a abordagem tradicional de ECMP. Estes resultados vão de acordo com os resultados obtidos em [1].

4.4.2 Topologia aleatória

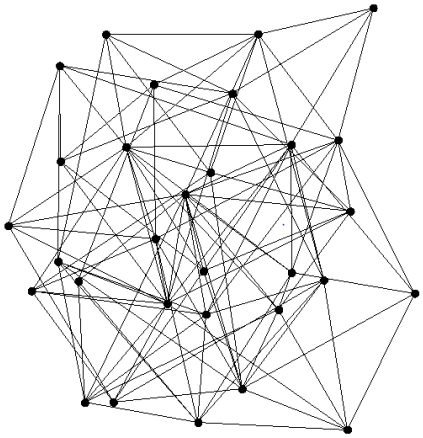

Figura 4.6: Topologia aleatória.

A segunda rede onde foram efetuadas simulações foi uma rede gerada aleatoriamente, que se encontra representada na figura 4.6. A rede é constituída por 30 nós onde cada nó tem uma probabilidade de 50% de se ligar a outro nó o que resultou num *degree* médio de 16. Numa rede gerada de forma aleatória é complicado definir nós de acesso e nós do *core*, como tal, tal como na rede Abilene todos os nós foram considerados nós de acesso.

Na figura 4.7 encontram-se os resultados teóricos obtidos, onde é possível observar que os métodos de encaminhamento baseados em álgebra são os que se espera que obtenham um pior desempenho. Estes resultados são validados pelos resultados obtidos através da aplicação SDN, que se encontram na figura 4.8, onde é possível ver que os resultados são semelhantes aos teóricos com exceção à abordagem baseada em álgebra onde os elementos

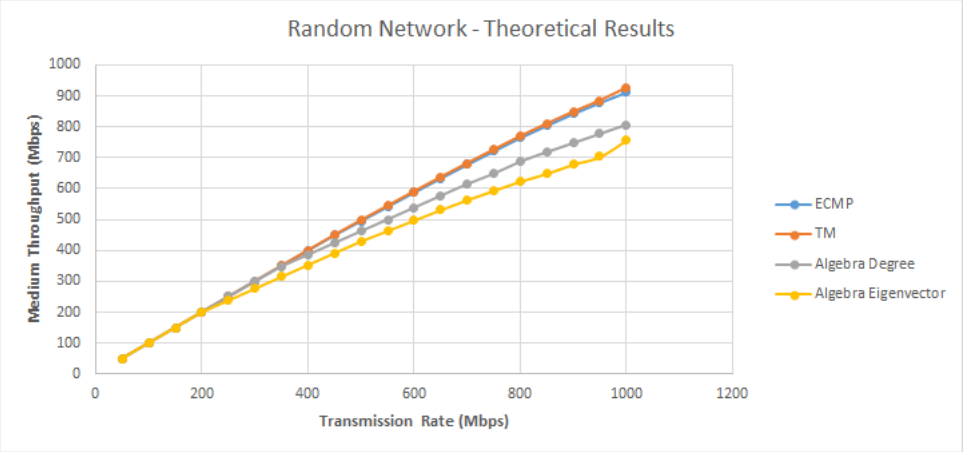

Figura 4.7: Resultados teóricos obtidos para a topologia gerada de forma aleatória.

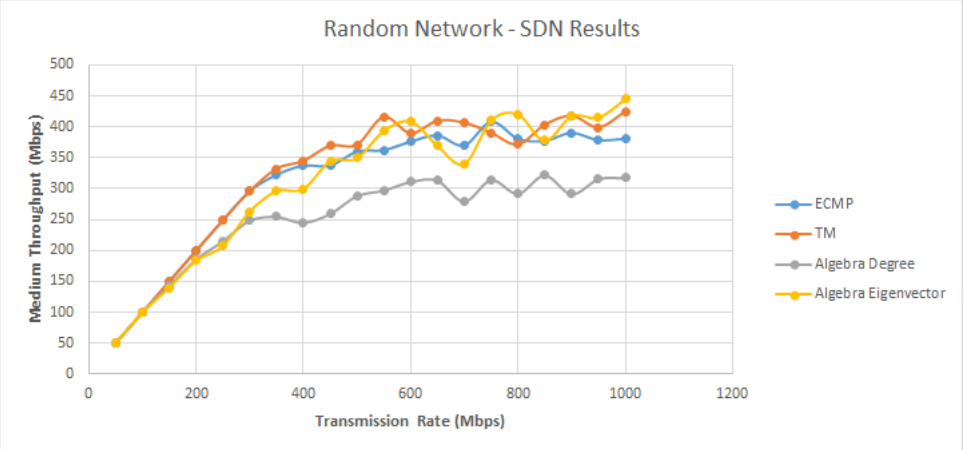

Figura 4.8: Resultados obtidos através da aplicação SDN para a topologia gerada de forma aleatória.

do conjunto S são distribuídos com base na *eigenvector centrality* dos nós. Os resultados obtidos para a Algebra Eigenvector resultam de o fato de apenas se estar a simular uma conexão entre os nós, onde a escolha dos caminhos foi a melhor. Contudo, espera-se que num cenário de convergência, os resultados sejam mais perto dos resultados teóricos.

Uma primeira análise à tabela 4.2 é possível verificar que os métodos de encaminhamentos que usam uma álgebra para o cálculo dos caminhos obtiveram uma quantidade de caminhos superior aos métodos que solucionam o problema do caminho mais curto. Com esta informação pressupõe-se que os métodos de encaminhamento algébricos irão ter uma melhor desempenho face aos restantes, devido a terem mais caminhos para distribuir o tráfego. Contudo, o número de caminhos calculados não é suficiente para se obter um melhor desempenho a nível de transporte de dados, pois o tráfego pode estar distribuído de forma a obter mais situações de congestão.

Para se entender melhor os resultados obtidos é necessário recorrer às definições introduzidas no início desta secção referentes a caminhos diferentes e a caminhos disjuntos.

	Caminho mais curto	Algebra Degree	Algebra Eigenvector
Caminhos disjuntos	1548	609	585
Caminhos diferentes	566	2468	2826
Nº total de caminhos	2114	3077	3411

Tabela 4.2: Número de caminhos entre os nós de acesso obtidos na rede gerada de forma aleatória.

	Caminho mais curto	Algebra Degree	Algebra Eigenvector
Nº Médio de Ligações Partilhadas	3	11	21

Tabela 4.3: Número médio de ligações partilhadas pelos caminhos diferentes na rede gerada de forma aleatória.

Os dados apresentados 4.2 indicam o número de caminhos disjuntos e diferentes para cada abordagem de encaminhamento. No caso dos caminhos que têm usam um método de encaminhamento resolvendo o problema do caminhos mais curto cerca de 72.5% dos caminhos são disjuntos, por outro lado utilizando uma abordagem algébrica, no melhor cenário apenas cerca de 19.8% dos caminhos são disjuntos. Em adição a existirem mais caminhos diferentes nas abordagens algébricas, os caminhos diferentes das abordagens baseadas em álgebra partilham em médias mais ligações dos que os caminhos quando comparados com as abordagens que visam a solucionar o problema do caminho mais curto. Analisado a tabela 4.3 é possível verificar que o número de ligações partilhadas entre os os caminhos diferentes obtidos através das abordagens baseadas em álgebra variam, no entanto no método Algebra Degree os caminhos diferentes partilham cerca de quatro vezes mais ligações que os caminhos baseados no problema do caminho mais curto e no caso da Algebra Eigenvector partilham sete vezes mais ligações.

Comparando os métodos de encaminhamento de caminho mais curto, existe uma pequena vantagem do método TM face a ECMP tradicional, onde para uma taxa de transmissão de 1000 megabits por segundo se espera que a método TM consiga transportar cerca de mais 1.5% que a abordagem ECMP. Uma abordagem utilizando ECMP tradicional resulta bem neste tipo de rede pois como o *degree* médio é muito alto existe uma grande variedade de caminhos entre os nós de acesso sendo que o bloqueio de caminhos e a centralidade apenas trazem um pequeno beneficio.

4.4.3 Topologia *power law*

A rede apresentada em 4.9 é outra rede que foi gerada e segue uma distribuição *power law*. Esta rede é constituída por 79 nós, onde o *degree* médio dos nós é de 20.15. Apesar de o *degree* médio ser de 20.15, a rede gerada tem valores de *degree* que variam imenso sendo que valor da mediana é 7 e cerca de 84.8% dos nós têm um *degree* menor ou igual a 12. Ou seja, existe um conjunto de nós que têm um *degree* muito elevado enquanto que a grande quantidade dos nós tem uma *degree* baixa. O caso mais real de uma rede que segue uma distribuição *power law* é a Internet, sendo que foi considerado que os nós de

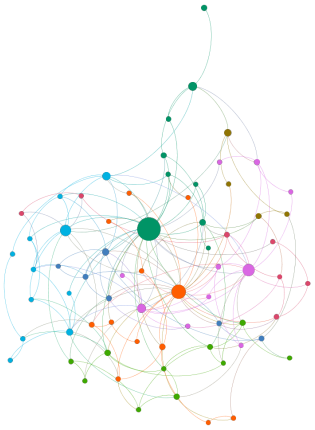

Figura 4.9: Topologia Power Law.

acesso são os nós com *degrees* mais baixos.

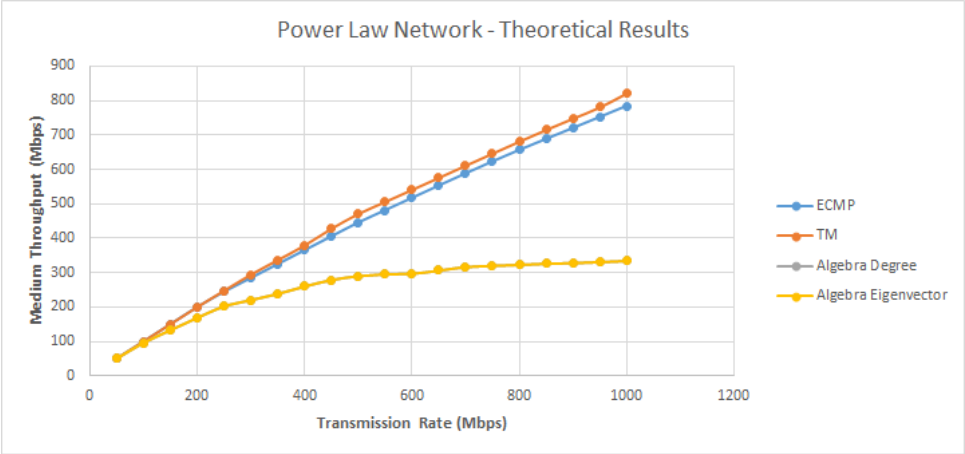

Figura 4.10: Resultados teóricos obtidos para a topologia *power law*.

A figura 4.10 e 4.11 mostram os resultados teóricos e os resultados obtidos através da aplicação SDN, sendo ambos novamente semelhantes. Tal como aconteceu para a rede gerada de forma aleatória, as abordagens baseadas em álgebra voltam a ter pior desempenho que as abordagens de caminho mais curto. Como é demonstrado na tabela 4.4, as abordagens baseadas em álgebra voltam a ter mais caminhos disponíveis para a distribuição do tráfego. Para esta rede, o número de caminhos aumento quase para o dobro. Contudo, o número de caminhos disjuntos é novamente muito menor que no caso de encaminhamento baseado no caminho mais curto. Os dados apresentados na 4.4 mostra que nas abordagens algébricas apenas cerca de 4.3% dos caminhos são caminhos disjuntos face aos cerca de 27.16% dos métodos que visam a solucionar o problema caminho mais curto.

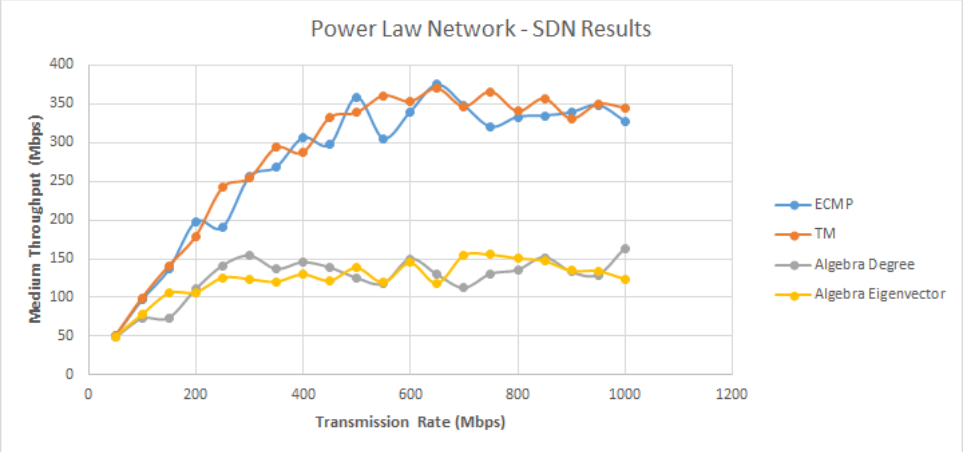

Figura 4.11: Resultados obtidos através da aplicação SDN para a topologia *power law*.

	Caminho mais curto	Algebra Degree	Algebra Eigenvector
Caminhos disjuntos	868	313	313
Caminhos diferentes	2328	6917	6917
Nº total de caminhos	3196	7230	7230

Tabela 4.4: Número de caminhos entre os nós de acesso obtidos na rede *power law*.

	Caminho mais curto	Algebra Degree	Algebra Eigenvector
Nº Médio de Ligações Partilhadas	8	41	41

Tabela 4.5: Número médio de ligações partilhadas pelos caminhos diferentes na rede *power law*.

Tal como aconteceu anteriormente, o número de ligações partilhadas pelos os caminhos diferentes obtidos nas abordagens algébricas é superior face às abordagens tradicionais de caminho mais curto. De acordo com os dados da tabela 4.5, os caminhos diferentes dos métodos de encaminhamento baseados em álgebra partilha cinco vezes mais ligações que os caminhos diferentes obtidos solucionando o problema do caminho curto. Aliando os dados da tabela 4.5 aos dados da tabela 4.4 é possível compreender o porquê de as abordagens algébricas terem um pior desempenho que as abordagens que viam a solucionar o problema do caminho mais curto.

Analisado apenas as abordagens de encaminhamento mais curto, o método TM volta a ter melhor desempenho que a abordagem tradicional de ECMP. A nível teórico, espera-se que quando os nós se encontram a transmitir a 1000 megabits por segundo o método TM consiga transportar cerca de mais 4.8% de tráfego.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões

O objetivo do trabalho apresentada nesta dissertação era conseguir estudar novos métodos de encaminhamento que conseguissem utilizar melhor os recursos das redes e conseguir transportar mais dados que as soluções tradicionais de encaminhamento. Para tal foram necessários várias etapas que se encontram resumidas de seguida.

No segundo capítulo começa-se por apresentar as soluções tradicionais de TE, apresentado as vantagens e desvantagens que levam a que não consigam satisfazer os requisitos atuais. A única maneira de se conseguir melhorar o desempenho da rede e influenciar a distribuição do tráfego é otimizar os custos das várias ligações de modo a obter um conjunto de caminhos que maximize a utilização dos recursos da rede, contudo obter uma solução ótima é muito complexo e no caso de se usar SDN traduz-se na implementação de muitas *flow-entries* que são um recurso crítico para os *switches* OpenFlow.

De seguida descreve-se o novo paradigma em arquitetura de redes, conhecido como SDN, enumerando as vantagens e desvantagens inerentes ao uso de SDN. É demonstrado que tirando partido das vantagens de SDN, tais como a visão global da rede, é possível desenvolver novas soluções de TE que permitam tomar decisões de encaminhamento que tomem em consideração o estado global da rede e façam uma melhor utilização da rede. Os problemas de escalabilidade que o uso de SDN introduz têm de ser abordados antes de se desenvolver uma aplicação de TE. De forma a contornar estes problemas é utilizada uma arquitetura de controlo hierárquica de dois níveis. Sendo descrito o porquê de se considerar uma arquitetura hierárquica de dois níveis e qual o impacto no desenvolvimento de soluções de TE. Finaliza-se o segundo capítulo explicando-se como se pode utilizar uma álgebra para modelar os problemas de encaminhamento e aumentar potencialmente o número de caminhos possíveis de se utilizar para realizar a distribuição

do tráfego.

No terceiro capítulo é apresentado o trabalho implementado. Foi desenvolvida uma ferramenta de modelação teórica que através de uma matriz de adjacência permite o estudo de forma analítica uma rede e permite o estudo de novos métodos de encaminhamento. A ferramenta foi desenvolvida de forma modular para permitir estudar os métodos de encaminhamento a vários níveis. Ao nível do cálculo dos caminhos estudou-se uma abordagem algébrica com o objetivo de potencialmente se conseguir aumentar o número de caminhos entre os nós de acesso da rede. Ao nível da distribuição do tráfego recorreu-se a uma métrica de grafos conhecida como *edge betweenness centrality* para calcular a quantidade de tráfego que deverá circular em cada caminho. A ferramenta teórica realiza uma simulação sob a rede emulada que serve como indicador de desempenho do método de encaminhamento estudado. Foi também desenvolvida uma aplicação SDN de forma a validar os resultados teóricos obtidos. A aplicação desenvolvida não contém a inteligência do cálculo dos caminhos, sendo que lê os caminhos de um ficheiro criado pela ferramenta de modelação teórica.

Finalmente, no capítulo 4 apresenta-se os cenários de simulação considerados e os resultados obtidos. Realizaram-se simulações em três redes, uma rede de *backbone* e duas redes geradas. Numa primeira abordagem é possível verificar que o uso de uma álgebra pode aumentar o número de caminhos entre os nós de acesso de uma rede. Contudo, o número de caminhos a utilizar não é um indicador suficiente para melhorar o desempenho dos métodos de encaminhamento. As experiências demonstraram que para além do número de caminhos é necessário analisar a quantidade de caminhos diferentes e caminhos disjuntos, e analisar o número de ligações partilhadas entre os caminhos diferentes dos diferentes métodos de encaminhamento. O aumento do número de caminhos onde existe uma diminuição de caminhos disjuntos demonstrou ter pior desempenhos que as abordagens tradicionais que visam solucionar o problema do caminho mais curto.

Outro aspeto relevante é a forma como são distribuídos os elementos do conjunto S da álgebra, tendo sido distribuídos recorrendo a métricas de grafos, nomeadamente o *degree* dos nós e *eigenvector centralties*. A distribuição dos elementos do conjunto S demonstrou ter impacto tanto na quantidade de dados transportados na rede como no cálculo dos caminhos entre os nós de acesso.

Comparando as técnicas de encaminhamento que visam resolver o problema do caminho mais curto, verificou-se que o uso da *betweenness centrality* das ligações conjugado com o bloqueio de caminhos congestionados conseguem obter um melhor desempenho quando comparada com ECMP tradicional. Demonstrando que mesmo para soluções de encaminhamento baseadas no caminho mais curto ainda existe espaço para melhoria.

Os resultados teóricos foram comparados com os resultados obtidos através da aplicação SDN. Em grande parte dos resultados, os resultados obtidos foram semelhantes e quando não o foram a explicação recai no fato de apenas se ter simulado uma vez a transmissão de dados na aplicação SDN. No caso dos resultados teóricos é o que se espera

obter numa situação de convergência, visto que como é uma simulação teórica é possível dividir o tráfego pelas os diferentes caminhos sem ter em conta as limitações reais. Em adição o algoritmo de controlo de tráfego pode ser diferente do usado pelo mininet. Esta validação de resultados indica que se pode usar a ferramenta de modelação teórica para estudar novos mecanismos de encaminhamentos de forma simples e rápida antes de se desenvolver novas aplicações SDN, pois os resultados obtidos encontram-se perto da realidade.

5.2 Trabalho futuro

Após a conclusão do trabalho feito nesta dissertação, foi encontrado espaço para melhorias. Algumas das possibilidades são descritas de seguida.

5.2.1 Cálculo de caminhos de menos preferidos

Uma hipótese que pode levar a uma melhor utilização dos recursos da rede é quando todos os caminhos igualmente preferidos estiverem congestionados existe um novo cálculo de caminhos. Este novo cálculo irá calcular os próximos caminhos preferidos e distribuir o tráfego por esses mesmos caminhos. Esta abordagem pode potencialmente aumentar o transporte de dados nas redes.

5.2.2 Uso de *Machine Learning*

Com a introdução do conceito de *Machine Learning* em SDN é possível prever o tráfego que irá circular e desta forma obter uma solução de encaminhamento ótima ou perto, de uma maneira simples. Acomodando a rede aos requisitos de tráfego irá aumentar a quantidade de tráfego transportada na rede.

5.2.3 Álgebras especificas para cada rede

Apesar dos resultados negativos obtidos, o uso de uma álgebra em alguns cenários conseguiu aumentar o número de caminhos possíveis para distribuir o tráfego entre os nós de acesso. Esta cenário abre espaço para o surgimento de novas álgebras que que sejam especificas para certas redes e que possam trazer benefícios.

Bibliografia

[1] P. Amaral, L. Bernardo e P. F. Pinto. "Scaling SDN based Traffic Engineering with a low complexity approach". Em: (2015).

[2] E. Rosen, A. Viswanathan e R. Callon. *Multiprotocol Label Switching Architecture*. URL: https://www.rfc-editor.org/rfc/rfc3031.txt (acedido em 04/02/2016).

[3] A. Pathak, M. Zhang e Y. Hu. "Latency inflation with mpls-based traffic engineering". Em: *ACM SIGCOMM IMC* (2011), pp. 463–472.

[4] C. Hong, S. Kandula e R. Mahajan. "Achieving high utilization with software-driven WAN". Em: *Sigcomm 2014* (2013), pp. 15–26.

[5] B. Fortz e M. Thorup. "Increasing Internet Capacity Using Local Search". Em: *Computational Optimization and Applications* 29.1 (2004), pp. 13–48.

[6] D. Xu, M. Chiang e J. Rexford. "Link-State Routing With Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering". Em: *IEEE/ACM Transactions on Networking* 19.6 (2011), pp. 1717–1730.

[7] N. Michael, A. Tang e D. Xu. "Optimal Link-state Hop-by-hop Routing". Em: *Icnp* (2013), pp. 1–10.

[8] N. McKeown. *How SDN will Shape Networking*. 2011. URL: https://www.youtube.com/watch?v=c9-K5O_qYgA (acedido em 08/01/2016).

[9] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmoly e S. Uhlig. "Software-Defined Networking : A Comprehensive Survey". Em: *Proceedings of the IEEE* 103.1 (2015), pp. 14 –76.

[10] M. Jammal, T. Singh, A. Shami, R. Asal e Y. Li. "Software defined networking: State of the art and research challenges". Em: *Computer Networks* 72 (2014), pp. 74–98.

[11] S. Agarwal. "Traffic engineering in software defined networks". Em: *2013 Proceedings IEEE*. 2013, pp. 2211–2219.

[12] *OpenFlow*. URL: https://www.opennetworking.org/sdn-resources/openflow (acedido em 09/01/2016).

[13] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt e T. Gayraud. "Software-Defined Networking: Challenges and research opportunities for Future Internet". Em: *Computer Networks* 75 (2014), pp. 453–471.

[14] F. Hu, Q. Hao e K. Bao. "A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation". Em: *IEEE Communications Surveys & Tutorials* 16.c (2014).

[15] M. K. Shin, K. H. Nam e H. J. Kim. "Software-defined networking (SDN): A reference architecture and open APIs". Em: *International Conference on ICT Convergence* (2012), pp. 360–361.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, Others e S. Shenker. "Onix: A distributed control platform for large-scale production networks". Em: *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (2010), pp. 1–6.

[17] *NOX*. URL: https://github.com/noxrepo/nox (acedido em 09/01/2016).

[18] *POX*. URL: https://github.com/noxrepo/pox (acedido em 09/01/2016).

[19] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado e R. Sherwood. "On controller performance in software-defined networks". Em: *Proceeding Hot-ICE'12 Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services* (2012), pp. 10–10.

[20] *Beacon*. URL: https://openflow.stanford.edu/display/Beacon/Home (acedido em 09/01/2016).

[21] *Floodlight*. URL: http://www.projectfloodlight.org/floodlight/ (acedido em 09/01/2016).

[22] SDxCentral. *List of SDN Controller Vendors*. URL: https://www.sdxcentral.com/resources/sdn/sdn-controllers/sdn-controllers-comprehensive-list/ (acedido em 08/01/2016).

[23] *POF*. URL: http://www.poforwarding.org/ (acedido em 09/01/2016).

[24] A. Doria. *Forwarding and Control Element Separation*. 2010. URL: http://www.ietf.org/rfc/rfc5810.txt.

[25] *OVSDB*. URL: http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf (acedido em 09/01/2016).

[26] *OpFlex*. URL: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html (acedido em 09/01/2016).

[27] *Open State*. URL: http://openstate-sdn.org/ (acedido em 09/01/2016).

[28] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner e S. Louis. "OpenFlow: Enabling Innovation in Campus Networks". Em: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[29] ONF. *OpenFlow Switch v1.5.1*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf (acedido em 09/01/2016).

[30] *Switch Light*. URL: http://www.bigswitch.com/products/switch-light (acedido em 09/01/2016).

[31] *ofsoftswitch13*. URL: https://github.com/CPqD/ofsoftswitch13 (acedido em 09/01/2016).

[32] *Open vSwitch*. URL: http://openvswitch.org/ (acedido em 09/01/2016).

[33] *Pica8*. URL: http://www.pica8.com/ (acedido em 09/01/2016).

[34] M. Casado. *OpenStack and Network Virtualization*. 2013. URL: http://blogs.vmware.com/vmware/2013/04/openstack-and-network-virtualization.html (acedido em 09/01/2016).

[35] *NS-3*. URL: https://www.nsnam.org/ (acedido em 09/01/2016).

[36] J. Pelkey. *OpenFlow Software Implementation Distribution*. URL: http://code.nsnam.org/jpelkey3/openflow (acedido em 09/01/2016).

[37] *Mininet*. URL: http://mininet.org/ (acedido em 09/01/2016).

[38] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer e J. Zhou. "B4: Experience with a Globally-Deployed Software Defined WAN". Em: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13* (2013), p. 12.

[39] S. H. Yeganeh, A. Tootoonchian e Y. Ganjali. "On scalability of software-defined networking". Em: *IEEE Communications Magazine* 51.2 (2013), pp. 136–141.

[40] A. A. R. Curtis, J. C. J. J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma e S. Banerjee. "DevoFlow: scaling flow management for high-performance networks". Em: *Proceedings of the {ACM} {SIGCOMM} 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, {ON}, Canada, August 15-19, 2011* (2011), pp. 254–265.

[41] A. Tootoonchian e Y Ganjali. "Hyperflow: a distributed control plane for openflow". Em: *Proceedings of the 2010 internet network ...* (2010).

[42] J. Sobrinho. "An algebraic theory of dynamic network routing". Em: *IEEE/ACM Transactions on Networking* 13.5 (2005), pp. 1160–1173.

[43] M. Gondran e M. Minoux. *Graphs, Dioids and Semirings*. Vol. 41. 2008.

[44] P. Amaral. "Multipath inter - domain policy routing". Phd thesis. Universidade Nova de Lisboa, 2012.

[45] P. Amaral, L. Bernardo e P. Pinto. "Achieving correct hop-by-hop forwarding on multiple policy-based routing paths". Em: (2015).

[46] J. S. Baras e G. Theodorakopoulos. *Path Problems in Networks*. Vol. 3. 1. 2010, pp. 1–77.

[47] T. G. Griffin e A. J. Gurney. "Increasing Bisemigroups and Algebraic Routing". Em: *Lecture Notes in Computer Science* 4988/2008 (2008), pp. 123–137.

[48] J. L. Sobrinho e T. G. Griffin. "Routing in equilibrium". Em: *19th Intenational Symposium on Mathematical Theory of Networks and Systems* (2010), pp. 1–7.

[49] A. J. Gurney. "Construction and Verification of Routing Algebras". Phd Thesis. University of Cambridge, 2009, pp. 1–150.

[50] T. G. Griffin, F. B. Shepherd e G. Wilfong. "The stable paths problem and interdomain routing". Em: *IEEE/ACM Transactions on Networking (TON)* 10.2 (2002), pp. 232–243.

[51] *igraph library*. URL: http://igraph.org/python/ (acedido em 21/07/2016).

[52] *Breadth-First Search*. URL: https://en.wikipedia.org/wiki/Breadth-first_search (acedido em 27/07/2016).

[53] M. Girvan e M. E. J Newman. "Community structure in social and biological networks". Em: (2001), pp. 1–8.

[54] B. Towles e W. Dally. "Worst-case Traffic for Oblivious Routing Functions". Em: *IEEE Computer Architecture Letters* 1.1 (2002), pp. 1–4.

[55] *Munkres implementation for Python*. URL: https://github.com/bmc/munkres (acedido em 14/08/2016).

[56] *Hungarian Algorithm*. URL: https://en.wikipedia.org/wiki/Hungarian_algorithm (acedido em 14/08/2016).

[57] *Mininet Overview*. URL: http://mininet.org/overview/ (acedido em 14/08/2016).

[58] *Iperf*. URL: https://iperf.fr/ (acedido em 14/08/2016).

[59] *Computer Networking: Principles, Protocols and Practice*. URL: http://cnp3book.info.ucl.ac.be/2nd/html/exercises/network.html (acedido em 14/08/2016).

[60] *Abilene Network*. URL: https://en.wikipedia.org/wiki/Abilene_Network (acedido em 14/08/2016).

[61] *Internet2 Community*. URL: http://www.internet2.edu/ (acedido em 14/08/2016).