Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Alisson Cavalcante e Silva

Balanceamento de carga entre caminhos utilizando redes definidas por software

Rio de Janeiro
2020

Alisson Cavalcante e Silva

Balanceamento de carga entre caminhos utilizando redes definidas por software

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Orientador: Prof. D.Sc. Marcelo Gonçalves Rubinstein

Rio de Janeiro

2020

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

S586 Silva, Alisson Cavalcante e.
Balanceamento de carga entre caminhos utilizando redes definidas por software / Alisson Cavalcante e Silva. – 2020.
75f.

Orientadores: Marcelo Gonçalves Rubinstein.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia eletrônica - Teses. 2. Redes de computadores - Teses. 3. Roteadores (Redes de computadores) - Teses. 4. Ethernet (Redes locais de computadores) - Teses. I. Rubinstein, Marcelo Gonçalves. II. Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia. III. Título.

CDU 004.71

Bibliotecária: Júlia Vieira – CRB7/6022

_____ _____

Assinatura Data

Alisson Cavalcante e Silva

Balanceamento de carga entre caminhos utilizando redes definidas por software

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Aprovada em 18 de fevereiro de 2020.
Banca Examinadora:

Prof. D.Sc. Marcelo Gonçalves Rubinstein (Orientador)
PEL/UERJ

Prof. Dr. Luis Henrique Maciel Kosmalski Costa
PEE/COPPE/UFRJ

Prof. D.Sc. Rodrigo de Souza Couto
PEE/COPPE/UFRJ

Rio de Janeiro
2020

DEDICATÓRIA

Aos meus filhos Felipe e Alice, que este feito os inspire a caminhar sempre mais longe. À minha esposa Monique, aos meus pais Ozias e Jane e à minha sogra Maria do Carmo. Todos vocês contribuíram imensamente para que esta jornada fosse concluída com sucesso.

AGRADECIMENTOS

Você pode sim chegar ao seu destino. Mas jamais o fará sozinho.

Em primeiro lugar, toda minha gratidão seja dada ao Autor da vida, o Senhor Deus.

Muito carinho e agradecimento à minha esposa Monique, pela compreensão dos muitos momentos que poderíamos passar juntos, mas que foram empregados nesta caminhada. "Nunca pare de nadar!"

Agradeço à Marinha do Brasil pela oportunidade de aperfeiçoamento profissional e pessoal.

Ao Programa de Pós Graduação em Engenharia Eletrônica da Universidade do Estado do Rio de Janeiro, por aceitar meu ingresso em seu curso de Mestrado.

Ao Prof. Marcelo Gonçalves Rubinstein, pelo conhecimento, orientação, incentivo e confiança em mim depositados.

Aos Professores Alexandre Sztajnberg, Francisco Figueiredo Goytacaz Sant'Anna e Rodrigo de Souza Couto, pelo apoio e ensinamentos transmitidos.

Aos membros da banca examinadora pela disponibilidade e aceite em avaliar este trabalho.

Ao Comandante Humberto Ferreira Ramos Junior, por ter acreditado e contribuído para que o meu ingresso neste curso se tornasse uma realidade.

Aos amigos que fiz no dia a dia de convivência na Universidade do Estado do Rio de Janeiro.

E a todos que não foram citados, mas que contribuíram para que essa jornada fosse concluída com empenho.

A gravidade explica os movimentos dos planetas, mas não pode explicar quem colocou os planetas em movimento. Deus governa todas as coisas e sabe tudo que é ou que pode ser feito.

Isaac Newton

RESUMO

SILVA, Alisson Cavalcante e. *Balanceamento de carga entre caminhos utilizando redes definidas por software*. 2020. 75 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2020.

Na última década, o volume de dados produzido e armazenado em escala global aumentou significativamente. Na busca em manter o consumo de informação e de serviços ocorrendo de forma ininterrupta, as empresas estão investindo em enlaces redundantes com o propósito de diminuir as chances de uma indisponibilidade do acesso à Internet. Assim, dispondo de enlaces redundantes é possível utilizá-los não somente durante as situações de indisponibilidade, mas como também no dia a dia realizando balanceamento de carga entre eles. O balanceamento de carga entre enlaces pode ser realizado utilizando roteamento por multicaminhos. Assim, os fluxos de dados podem ser distribuídos simultaneamente por mais de um caminho existente entre origem e destino. Contudo, algumas redes como a *Ethernet* utilizam o modelo de encaminhamento de pacotes de caminho único baseado no *Spanning Tree Protocol* (STP) para evitar *loops* na rede, em situações que múltiplos comutadores *Ethernet* são interligados entre si. Tal característica não permite a utilização de enlaces ociosos para diminuir o congestionamento do caminho e aumentar a largura de banda agregada da rede. Como solução, este trabalho apresenta uma proposta de mecanismo de balanceamento de carga entre caminhos utilizando redes definidas por software (SDNs - *Software Defined Networks*). O mecanismo proposto, denominado MLB (*Multipath Load Balance*), é baseado em um mecanismo de balanceamento de carga usando redes SDN proposto na literatura. Porém, diferentemente desse mecanismo da literatura, este realiza computação de caminhos com enlaces disjuntos e conta com um controle de comutação que verifica se a ocupação atual do caminho ultrapassa 50% de sua capacidade e se o potencial novo caminho computado apresenta uma ocupação pelo menos 10% menor do que a do caminho atual, de forma a não realizar a troca de caminhos sem que haja um ganho razoável. Sendo assim, como forma de avaliar o funcionamento do mecanismo proposto, este trabalho também apresenta uma avaliação de desempenho que compara o funcionamento dos dois mecanismos com o modo de funcionamento padrão do controlador de SDN OpenDaylight (ODL). Os resultados obtidos mostram que com uso do MLB foi possível aumentar em 95% o valor da largura de banda agregada e diminuir em cerca de 44,2% a perda de pacotes em comparação ao modo de funcionamento padrão do ODL.

Palavras-chave: SDN; Balanceamento de carga; Roteamento multicaminhos.

ABSTRACT

SILVA, Alisson Cavalcante e. *Load balancing between paths using software defined networks*. 2020. 75 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2020.

In the last decade, the volume of data produced and stored on a global scale has increased significantly. In the quest to keep the consumption of information and services occurring uninterruptedly, companies are investing in redundant links in order to reduce the chances of an unavailability of Internet access. Thus, having redundant links it is possible to use them not only during unavailability situations, but also on a daily basis balancing the load between them. Load balancing between links can be accomplished using multipath routing. Thus, data streams can be distributed simultaneously over more than one path between source and destination. However, some networks like Ethernet use the single path packet forwarding model based on the Spanning Tree Protocol (STP) to avoid loops on the network, in situations where multiple switches Ethernet are interconnected. This feature does not allow the use of idle links to reduce path congestion and increase the aggregate network bandwidth. As a solution, this work presents a proposal for a load balancing mechanism between paths using software defined networks (SDNs). The proposed mechanism, called MLB (Multipath Load Balance), is based on a load balancing mechanism using SDN networks proposed in the literature. However, unlike this mechanism in the literature, it performs path computation with disjoint links and has a switching control that checks whether the current path occupation exceeds 50% of its capacity and whether the potential new computed path has an occupation at least 10% less than the current path, so as not to change paths without a reasonable gain. Therefore, as a way to evaluate the functioning of the proposed mechanism, this work also presents a performance evaluation that compares the functioning of the two mechanisms with the standard operating mode of the SDN OpenDaylight controller (ODL). The results obtained show that with the use of MLB it was possible to increase the value of the aggregated bandwidth by 95% and decrease packet loss by about 44,2% compared to the standard ODL operating mode.

Keywords: SDN; Load balancing; Multipath routing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
ATM	*Asynchronous Transfer Mode*
BFS	*Breadth First Search*
CAPEX	*CAPital EXpenditure*
FDDI	*Fiber-Distributed Data Interface*
ICMP	*Internet Control Message Protocol*
IDC	*International Data Corporation*
IoT	*Internet of Things*
IP	*Internet Protocol*
MAC	*Medium Access Control*
MLB	*Multipath Load Balance*
NOS	*Network Operating System*
ONF	*Open Networking Foundation*
OPEX	*OPerational EXpenditure*
OvS	*Open vSwitch*
SDN	*Software Defined Networks*
STP	*Spanning Tree Protocol*
SLA	*Service Level Agreement*
TE	*Traffic Engineering*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
VM	*Virtual Machine*
WAN	*Wide Area Network*
WRR	*Weighted Round-Robin*

SUMÁRIO

INTRODUÇÃO

Na última década, o volume de dados produzido e armazenado em escala global aumentou significativamente. Segundo estudo realizado pela International Data Corporation (IDC) (GANTZ; REINSEL; RYDNING, 2018), em 2010 o volume de dados ficava abaixo de 5 Zettabytes[1], chegando a 33 Zettabytes em 2018 e com projeção de atingir 175 Zettabytes até o ano de 2025. De fato, esse aumento de dados deve-se à crescente adesão de usuários aos serviços ofertados na Internet. Tarefas como leitura de *email*, negociações de *e-commerce*, monitoramento de transações financeiras, pagamentos de taxas e impostos via *Internet banking*, adesão a pregões eletrônicos, contratações de funcionários, acesso a mídias sociais, treinamentos *online* a distância, acesso a serviços de *streaming* de áudio e vídeo e muitas outras fazem parte da rotina de acesso diária dos bilhões de usuários da Internet.

Em um mundo globalizado, uma indisponibilidade no acesso, causada por uma quebra ou congestionamento de um enlace, não é facilmente aceita pelos usuários. Manter um cenário de alta disponibilidade, mesmo que para uma pequena rede de computadores, tornou-se uma prerrogativa básica, além de uma trabalhosa demanda para os administradores de redes (RAMDHANI; HERTIANA; DIRGANTARA, 2016).

Na busca em manter o consumo de informação e de serviços ocorrendo de forma ininterrupta, as empresas estão investindo em enlaces redundantes com o propósito de diminuir as chances de uma indisponibilidade do acesso à Internet. O mercado tem se mostrado bastante promissor com ofertas de enlaces com maiores capacidades e bem mais baratos, além da confecção de acordos de níveis de serviços (SLA - *Service Level Agreement*) que determinam prazos de recuperação que chegam bem perto de um cenário de alta disponibilidade. Assim, dispondo de enlaces redundantes é possível utilizá-los não somente durante as situações de indisponibilidade, mas também no dia a dia realizando balanceamento de carga entre eles. Desta forma, é possível aumentar a vazão do tráfego de dados combinando a largura de banda de dois ou mais caminhos da rede.

O balanceamento de carga entre enlaces pode ser realizado utilizando roteamento por multicaminhos. Assim, os fluxos de dados podem ser distribuídos simultaneamente por mais de um caminho existente entre origem e destino. Com essa forma de trabalho pode-se evitar que alguns enlaces da rede se tornem ociosos, garantindo assim um maior aproveitamento dos recursos existentes na rede (SINGH; DAS; JUKAN, 2015).

Nas redes de computadores tradicionais cada comutador trabalha de forma individual, ou seja, as decisões de encaminhamento são tomadas por cada um dos comutadores

[1] Unidade utilizada para medir quantidade de dados digitais. Um Zettabyte corresponde a 10^{21} bytes.

por onde o tráfego passa. Cada dispositivo possui seu próprio plano de controle e de dados. Em situações que exijam modificações nas configurações dos planos de controle de cada dispositivo da rede, essa característica pode trazer grandes dificuldades para os administradores de redes. Como exemplo, pode-se destacar a tarefa de se configurar uma rota de forma manual entre origem e destino que possuam pontos intermediários no caminho (FEAMSTER; REXFORD; ZEGURA, 2014).

Neste aspecto, as redes definidas por software (SDNs - *Software Defined Networks*), se apresentam como uma alternativa que permite separar os planos de controle e de dados de cada dispositivo da rede, proporcionando um gerenciamento mais flexível e centralizado. Com o uso deste tipo de rede, a configuração de uma rota nos comutadores da rede pode ser feita em questões de segundos por uma aplicação desenvolvida pelo próprio administrador de rede. A SDN possibilita o desenvolvimento de aplicações especializadas que podem aplicar regras específicas nos dispositivos da rede por meio de uma entidade centralizada denominada controlador SDN (KREUTZ et al., 2015). O uso da centralização facilita a análise comparativa das condições de cada dispositivo da rede, como também a configuração destes de forma simultânea (RAMDHANI; HERTIANA; DIRGANTARA, 2016).

Nos últimos anos, esse novo paradigma ganhou atenção da indústria e vários fabricantes passaram a desenvolver *switchs* de emprego comercial com suporte à interface de programação de aplicação (API - *Application Programming Interface*) OpenFlow (LARA; KOLASANI; RAMAMURTHY, 2014; FEAMSTER; REXFORD; ZEGURA, 2014). Empresas como Google, Facebook, Microsoft e Verizon adotaram esta tecnologia nas redes de núcleos (*backbones*). O Google, por exemplo, utiliza o SDN como solução para interconectar seus *datacenters* espalhados pelo mundo (KREUTZ et al., 2015). Com uma boa recepção da API OpenFlow pela indústria, programadores voltados ao desenvolvimento de aplicações de rede puderam criar aplicações como balanceadores de carga, *firewalls*, sistemas de detecção de intrusão, controles de acesso, *traffic shapers*, entre outros (FEAMSTER; REXFORD; ZEGURA, 2014).

Motivação e Objetivos

As redes *Ethernet* têm se mostrado um caso de sucesso há mais de 30 anos. Durante todos esses anos, surgiram tecnologias concorrentes, como: *token ring*, *Fiber-Distributed Data Interface* (FDDI) e *Asynchronous Transfer Mode* (ATM). Mas a *Ethernet* continuou a desenvolver-se e a crescer, de forma a conquistar uma posição dominante no mercado de redes locais (KUROSE; ROSS, 2014; MYERS; NG; ZHANG, 2004).

Entretanto, este tipo de rede utiliza o modelo de encaminhamento de pacotes de caminho único baseado no *Spanning Tree Protocol* (STP) para evitar *loops* na rede. Tal

característica não permite a utilização de enlaces ociosos para diminuir o congestionamento do caminho e aumentar a largura de banda agregada da rede. Porém, uma solução que pode contornar este problema é utilização de balanceamento de carga entre enlaces.

Este trabalho apresenta uma proposta de mecanismo de balanceamento de carga entre caminhos utilizando redes definidas por software, que substitui o paradigma formado pelo controle descentralizado com forte acoplamento do plano de controle e de dados existentes nas redes tradicionais, pelo paradigma de controle centralizado com separação do plano de controle e de dados.

O fato das redes SDN apresentarem uma arquitetura centralizada fornece ao controlador uma visão global da topologia de rede, o que facilita a computação de caminhos entre a origem e o destino. Outro ponto importante é que redes SDN são programáveis. Essa característica permite ao administrador de rede criar aplicações inteligentes, usando dados coletados da rede, que podem tomar decisões como, por exemplo, por qual caminho com menor taxa de ocupação um fluxo de rede deve ser encaminhado. Redes Ethernet utilizam o STP para gerar uma árvore de cobertura com caminho único, que faz com que o tráfego de dados seja enviado de um ponto a outro na rede sem que ocorram *loops* que podem ser gerados por caminhos redundantes (SINGH; DAS; JUKAN, 2015). Porém, o controlador de uma rede SDN, graças à sua visão global da rede, torna dispensável o uso do STP e assume a função de gerenciar os caminhos da rede.

O mecanismo de balanceamento de carga proposto, denominado MLB (*Multipath Load Balance*), é baseado no mecanismo de balanceamento de carga proposto por Nayan Seth (SETH, 2016; HASSAN, 2017). No entanto, o MLB difere do mecanismo de Seth por contar com uma função denominada "controle de comutação" que utiliza algumas premissas para realizar a troca de caminho, evitando que repetidas trocas acarretem um baixo desempenho. Este controle de comutação verifica se a ocupação atual do caminho ultrapassa 50% da capacidade deste e se o potencial novo caminho computado apresenta uma ocupação pelo menos 10% menor do que a ocupação do caminho atual. Além disso, o MLB realiza a computação de caminhos com enlaces disjuntos, enquanto que o mecanismo de Seth utiliza enlaces não disjuntos.

Além de apresentar a proposta de balanceamento de carga pelo mecanismo MLB, esta dissertação também avalia o desempenho através de experimentos com o emulador Mininet (LARA; KOLASANI; RAMAMURTHY, 2014; LANTZ; HELLER; MCKEOWN, 2010) do MLB, da implementação de Seth e do controlador ODL desprovido de balanceamento de carga. Os resultados obtidos mostram que o MLB apresentou melhores desempenhos na agregação da largura de banda (um aumento em 63%), na diminuição da perda pacotes (uma redução de 42%) e promoveu maior nível de justiça na disputa dos fluxos pela largura de banda.

Organização da Dissertação

Esta dissertação está organizada em um total de seis capítulos dispostos da seguinte forma. O Capítulo 1 apresenta características das redes de computadores tradicionais, expõe a abstração de planos de funcionalidade das redes de computadores, apresenta o conceito e a arquitetura das redes definidas por software, como também o controlador ODL e o comutador virtual Open vSwitch, ambos utilizados nos experimentos realizados neste trabalho. O Capítulo 2 apresenta alguns trabalhos publicados pela comunidade acadêmica relacionados com balanceamento de carga utilizando SDN. O Capítulo 3 apresenta a ideia de multicaminhos, o roteamento por multicaminhos e os seus três componentes: computação de caminhos, divisão de tráfego e seleção de caminhos; além de apresentar a teoria de fluxo máximo em redes, utilizada na computação de caminhos. O Capítulo 4 apresenta a biblioteca python "*NetworkX*" e os mecanismos de balanceamento de carga avaliados nesta dissertação. O Capítulo 5 detalha como foi realizada a avaliação de desempenho, as ferramentas utilizadas, as métricas e modelos de tráfegos utilizados e os resultados obtidos. E, finalmente a conclusão encerra este trabalho fornecendo argumento conclusivo baseado nos resultados alcançados e apontando uma diretriz para novos estudos sobre balanceamento de carga utilizando SDN.

1 REDES DE COMPUTADORES: REDES TRADICIONAIS E O NOVO PARADIGMA DE REDES DEFINIDAS POR SOFTWARE

Este capítulo tem por objetivo apresentar as características das redes de computadores tradicionais e do novo paradigma de redes definidas por software. A Seção 1.1 fornece uma alusão das características de funcionamento das redes de computadores tradicionais. Enquanto que a Seção 1.2 apresenta a ideia de plano de funcionalidades existentes nas redes de computadores. E concluindo o capítulo, são apresentadas as características e o funcionamento das redes definidas por software.

1.1 Redes de Computadores Tradicionais

As redes de computadores tradicionais foram projetadas inicialmente para funcionarem utilizando uma estrutura descentralizada e com forte acoplamento dos planos de controle e dados (explicados na próxima seção). Isso significa que cada dispositivo da rede possui tanto plano de dados quanto o plano de controle incorporados em seu núcleo, como pode-se ver na Figura 1. Tal estrutura se mostrou importante para garantir resiliência e bom desempenho ao funcionamento da rede. Porém, com o passar dos anos e o crescimento das redes, esse tipo de estrutura acabou se mostrando extremamente complexa para se gerenciar (KREUTZ et al., 2015). A estrutura descentralizada exige o uso de protocolos de roteamento que funcionam de forma distribuída e que exigem processamento de cada dispositivo da rede. Os dispositivos de rede precisam ser configurados separadamente gerando excessivo trabalho para os administradores da rede. Cada fabricante utiliza o seu próprio sistema operacional, com seus conjuntos de instruções e ferramentas de gerenciamento proprietárias, o que consolida uma arquitetura verticalmente integrada, estática e pouco flexível, dificultando grandes inovações. Isso tudo faz com que administradores de rede tenham que manter diferentes soluções de gerenciamento conforme novos equipamentos de diferentes fabricantes vão sendo adquiridos. Além disso, ainda precisam capacitar suas equipes para lidarem com os diferentes tipos de equipamentos que vão sendo incorporados à rede (KIM; FEAMSTER, 2013).

1.2 Planos de Funcionalidades das Redes de Computadores

Esta seção expõe o funcionamento das redes de computadores com base na divisão por três planos de funcionalidades segundo (KREUTZ et al., 2015):

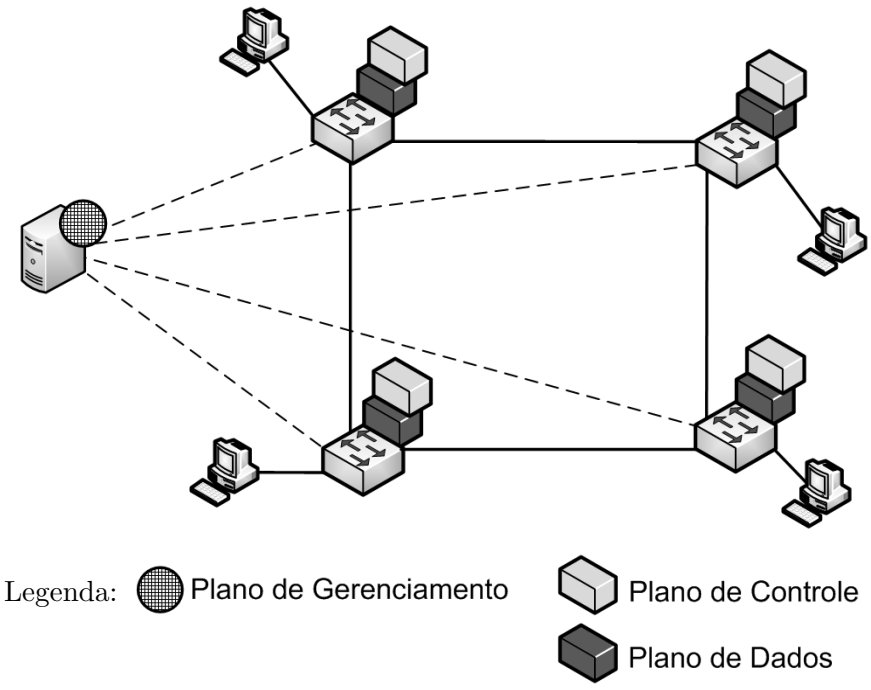
a) plano de dados: corresponde aos dispositivos da rede e possui a função

de encaminhar os dados entre as interfaces do dispositivo de rede. Neste são verificadas, por meio das correspondências de endereço MAC na tabela de comutação para os comutadores e endereço IP na tabela de roteamento para os roteadores, em quais interfaces os dados serão encaminhados;

b) plano de controle: onde está localizada a inteligência da rede. É responsável por definir as informações de controle que serão utilizadas pelo plano de dados ao encaminharem os pacotes. Representa os protocolos utilizados para preencher as tabelas de encaminhamento dos dispositivos de rede (tabela de comutação e tabela de roteamento);

c) plano de gerenciamento: É responsável por monitorar e configurar remotamente as funcionalidades do plano de controle. Neste são empregadas ferramentas de gerenciamento de rede, como por exemplo ferramentas baseadas no protocolo simples de gerenciamento de redes (SNMP - *Simple Network Management Protocol*), usado para monitorar e configurar remotamente dispositivos de rede.

Figura 1 - Visão da rede de computadores tradicional com plano de controle e de dados fortemente acoplado.



Contextualizando, o plano de gerenciamento define as políticas de rede, o plano de controle aplica as políticas definidas e o plano de dados encaminha os dados de acordo com as políticas aplicadas.

1.3 Redes Definidas por Software

As redes definidas por software trazem a promessa de permitir inovação na forma como se projetam e gerenciam as redes de computadores. Apesar de parecer que o termo SDN tenha surgido há apenas alguns anos atrás, a ideia de tornar as redes de computadores programáveis não é nova. Tal ideia surgiu nos meados dos anos 1990, junto com a popularização da Internet, por meio das redes ativas. Elas utilizavam uma API de rede que revelava recursos como por exemplo, processamento, fila de pacotes e armazenamento, existentes em cada um dos nós da rede e permitia a construção de funcionalidades personalizadas que podiam ser aplicadas aos subconjuntos de pacotes que trafegavam pelos nós (FEAMSTER; REXFORD; ZEGURA, 2014). No entanto, a falta de um problema convincente que justificasse o seu uso fez com que o projeto não evoluísse. Além disso, o roteadores das redes ativas não apresentavam bons desempenhos como os roteadores das redes tradicionais. No início dos anos 2000, a Internet ganhou popularidade e seu tráfego cresceu. As operadoras entenderam que precisavam de novas ferramentas para gerenciar suas redes, que os protocolos em uso até então não eram capazes de realizar tarefas mais complicadas, como controlar os caminhos utilizados pelo tráfego de rede e por isso não facilitavam a gerência do tráfego (FEAMSTER; REXFORD; ZEGURA, 2014). Pesquisadores da área de redes enxergaram que a solução encontrava-se em separar o plano de controle do plano de dados. Os fabricantes começaram a produzir equipamentos com a lógica de encaminhamento de dados separada do plano de controle. O que veio a seguir foi o surgimento de duas inovações: o uso de interfaces de comunicação entre o plano de controle e o plano de dados e uma arquitetura de controle logicamente centralizado (FEAMSTER; REXFORD; ZEGURA, 2014). Porém, cada fabricante adotava a sua própria interface, que muitas das vezes suportava apenas configurações de roteamento. Mas, em meados da década de 2010, pesquisadores da Universidade de Stanford desenvolveram uma interface aberta denominada de OpenFlow. Construída em cima de hardware *switchs*, esta nova interface permitiu que mais funções do que somente configurações de rota fossem realizadas (LARA; KOLASANI; RAMAMURTHY, 2014). Seguindo esta nova tendência, apareceram os controladores de redes SDN equipados com uma interface OpenFlow responsáveis por aplicar o plano de controle aos dispositivos de rede. Esta nova arquitetura transformou os comutadores e roteadores de pacotes em apenas encaminhadores de pacotes e toda lógica de controle ficou com os controladores de rede, ou sistemas operacionais de rede como também são conhecidos.

Na arquitetura de SDN, o controlador tem acesso a dados estatísticos coletados diretamente dos comutadores que o permitem tomar decisões de forma dinâmica. Os dispositivos utilizam tabelas de fluxos constituídas de regras de fluxos que estabelecem as ações de comutação ou descarte de pacotes, que os comutadores devem executar sempre que um fluxo estabelecer correspondência com uma destas regras. As aplicações de

rede podem utilizar os dados estatísticos para ajudá-las a desempenhar funções específicas na rede como *firewalls*, controles de acesso dinâmico, roteadores, balanceadores de carga, entre outros, inclusive realizando acúmulo de funções (MCKEOWN et al., 2008). E o fato dos dispositivos de rede poderem acumular funções ou serem programados com uma determinada função contribui para diminuir o capital investido (CAPEX - *CAPital EXpenditure*) com aquisição de novos equipamentos para compor a infraestrutura da rede. Além disso, o uso de programação de alto nível pelas aplicações de rede ajuda a diminuir as despesas operacionais (OPEX - *OPerational EXpenditure*) com treinamentos e especializações em soluções proprietárias desenvolvidas por cada fabricante de dispositivos, necessárias para que as equipes de rede continuem a manter a infraestrutura da rede operacional. Esses fatores tornam a SDN uma solução economicamente viável e atraente (COX et al., 2017).

1.3.1 A Arquitetura SDN

Este trabalho baseia-se na abstração proposta em (KREUTZ et al., 2015) para conceituar e definir a Arquitetura de SDN. Em seu trabalho, Kreutz apresenta quatro pilares que ajudam a definir a arquitetura de SDN:

a) desacoplamento dos planos de controle e dados: a função de controle é removida dos dispositivos, de forma a torná-los apenas simples encaminhadores de pacotes;

b) decisões de encaminhamento baseadas em fluxo e não em destino do pacote: um fluxo é definido por um conjunto de campos que formam o cabeçalho do pacote. Além disso, critérios de correspondência (filtros) são estabelecidos, de modo a dispararem ações (comutações ou descartes de pacotes);

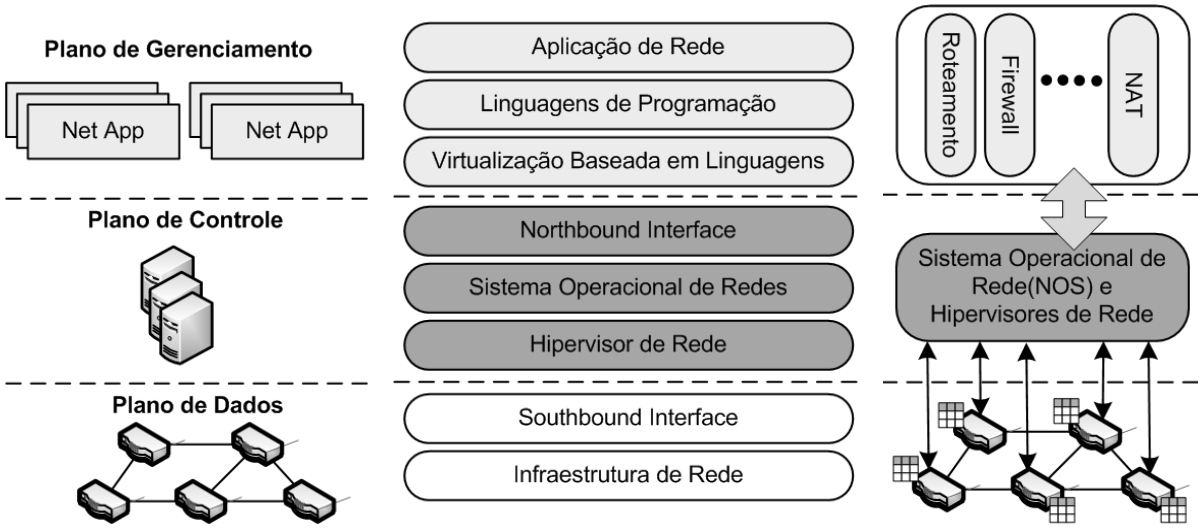
c) lógica de controle realizada por uma entidade externa: denominada controlador SDN, trata-se de uma plataforma de software que funciona em um servidor e fornece recursos e abstrações essenciais que facilitam a programação dos dispositivos de encaminhamento. Sua finalidade é semelhante à de um sistema operacional tradicional, por isso também é conhecida como sistema operacional de rede (NOS - *Network Operating System*);

d) programação da rede realizada por aplicações de rede: considerada a fundamental característica da SDN, ela define que as aplicações de rede

sejam executadas sobre o NOS, cabendo-lhe a tarefa de interagir com os dispositivos que fazem parte do plano de dados.

A centralização lógica do plano de controle em uma entidade externa com conhecimento global da rede simplifica o desenvolvimento de funções, serviços e aplicações de rede. E o fato de se utilizar linguagens de alto nível (aplicações de rede) para modificar as políticas de rede deixa o processo de manutenção das redes mais simples e menos propenso a erros quando comparado com a forma de executar as configurações específicas de baixo nível, comumente empregadas nos dispositivos de rede separadamente e via linha de comando (KREUTZ et al., 2015).

Figura 2 - Diferentes abstrações para a Arquitetura SDN.



(a) Planos (b) Camadas (c) Sistemas

Fonte: Versão de (COUTO, 2019) adaptada de (KREUTZ et al., 2015).

Em (KREUTZ et al., 2015), os autores apresentam a arquitetura SDN a partir de três diferentes visões, como mostrado na Figura 2.

A Figura 2(a) expõe a visão orientada em planos de funcionalidade em uma rede SDN. Nesta é possível ver que o plano de dados permanece dentro de cada dispositivo de encaminhamento da rede, enquanto que o plano de controle é separado deste em uma entidade externa, como por exemplo em um servidor de rede. O plano de gerenciamento é composto pelas aplicações de redes que podem ser executadas no mesmo servidor que hospeda o plano de controle ou em um servidor ou estação de trabalho remoto.

Já a Figura 2(b) apresenta uma abstração da arquitetura SDN dividida em oito diferentes camadas. De baixo para cima, a primeira camada é a de Infraestrutura de Rede, composta por dispositivos de rede parecidos com os existentes nas redes tradicionais, como: *switchs*, roteadores, *firewalls*, entre outros. Esta camada é similar a uma camada

do modelo de rede de computadores tradicional. A diferença é que estes dispositivos não possuem plano de controle localizado em seu núcleo. Eles possuem apenas o plano de dados composto por tabelas de fluxo que definem como os pacotes existentes em um fluxo serão tratados.

A segunda camada denominada *Southbound Interface* ou *Southbound API* consiste de uma API responsável por permitir a comunicação entre o NOS e os dispositivos de rede. Dentre as APIs existentes no mercado, a OpenFlow é mais amplamente aceita e implementada para SDN, sendo considerada um padrão de *Southbound API*.

A terceira camada é um hipervisor de rede que possibilita a virtualização de uma rede SDN. Assim, é possível que mais de um controlador possa utilizar a mesma rede física. Os recursos de hardware disponíveis na camada de infraestrutura podem ser utilizados entre múltiplos planos de controle.

Na quarta camada encontra-se o sistema operacional de rede. Seu papel é fornecer abstrações, serviços essenciais e APIs comuns aos desenvolvedores, para que a comunicação entre as aplicações de rede (escritas em linguagem de alto nível) e os dispositivos de rede possa acontecer de maneira simples. Funcionalidades genéricas como informações da topologia de rede e estado de rede, descoberta de dispositivos e configuração de rede são fornecidas como serviços do NOS.

A quinta camada apresenta a *Northbound Interface* ou *Northbound API* que consiste em uma API responsável por permitir a comunicação entre a linguagem de programação, na qual a aplicação de rede foi desenvolvida, e o controlador. Diferentemente do que ocorre com a *Southbound Interface*, a *Northbound Interface* não é padronizada; o que significa que os controladores podem utilizar diferentes *Northbound API* existentes, como por exemplo, ad hoc APIs, RESTful APIs, NETCONF, entre outras.

A sexta camada apresenta a Virtualização Baseada em Linguagens. Ela permite diferentes níveis de abstração dos componentes de hardware; como por exemplo, comutadores distribuídos podem ser manipulados como um único comutador. Um exemplo de linguagem que permite desenvolver este nível de abstração é a Pyretic (KREUTZ et al., 2015).

A sétima camada representa uma abstração para as linguagens de programação de alto nível, como Java e Python, que podem ser ferramentas poderosas como meio de implementar e fornecer abstrações para diferentes propriedades e funções para a rede, em substituição ao antigo modelo de linguagens de baixo nível utilizado para configurar os dispositivos de rede.

Na última camada de aplicação de rede encontram-se os aplicativos de rede, que podem ser vistos como os "cérebros da rede". Eles são responsáveis por implementar a política de controle que será traduzida para comandos (de baixo nível) que serão instalados nos dispositivos de rede localizados na camada de infraestrutura (plano de dados).

Por fim, a Figura 2(c) exibe uma visão sob a perspectiva da interação ente sistemas.

No topo da figura estão as aplicações que definem e entregam as políticas de rede ao NOS, para que este aplique regras de fluxos aos dispositivos de rede que serão responsáveis por estabelecerem correspondências com os fluxos da rede.

1.3.2 OpenFlow

O OpenFlow nasceu na Universidade de Stanford da necessidade de pesquisadores da área de redes de poderem testar protocolos experimentais nas redes acadêmicas utilizando comutadores de pacotes Ethernet comerciais (*commercial Ethernet switches*) (MCKEOWN et al., 2008). Inicialmente, tinha o objetivo de incentivar os fabricantes de dispositivos de rede a adicionarem o OpenFlow em seus comutadores para que estes pudessem ser empregados no núcleo central da rede (*backbone*) acadêmica de Stanford. Assim, seria possível atender as necessidades da comunidade de pesquisa que procurava ativamente maneiras de conduzir trabalhos experimentais em arquiteturas de rede "limpas", dentro de um ambiente operacional e favorável à inovação. Além disso, os fabricantes não precisavam expor o funcionamento interno de seus comutadores (FEAMSTER; REXFORD; ZEGURA, 2014).

Com o passar do tempo, o OpenFlow chegou às redes de *datacenters* e os fabricantes sentiram a necessidade de adotá-lo em seus servidores e *switchs*, de maneira a proporcionar o aumento das funcionalidades da rede, a redução da complexidade dos dispositivos de rede, como também continuarem se mantendo em um mercado extremamente competitivo. A *Open Networking Foundation* (ONF) acabou padronizando o OpenFlow e este veio a se tornar um dos principais padrões de *Southbound Interface* para redes SDN. Tudo isso faz com que o OpenFlow seja reconhecido não somente como um protocolo de comunicação, mas também como um modelo de arquitetura para dispositivos de rede (STALLINGS, 2013).

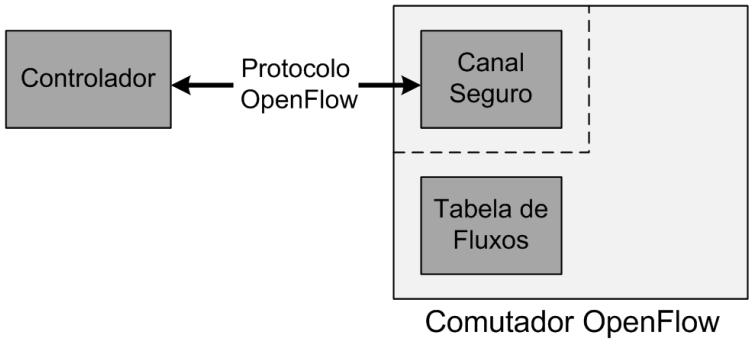
Segundo (LARA; KOLASANI; RAMAMURTHY, 2014), a arquitetura do OpenFlow, apresentada na Figura 3, é composta por três componentes principais:

a) controlador: é responsável por manipular a tabela de fluxo do comutador e coletar informações dos contadores destas, usando o protocolo OpenFlow;

b) canal seguro: trata-se de interface que permite a comunicação do controlador com o comutador OpenFlow. Por meio deste canal, o controlador gerencia os comutadores e consegue receber e enviar pacotes para os comutadores;

c) comutador OpenFlow: compatível com o protocolo OpenFlow, utiliza tabelas de fluxo para encaminhar os pacotes da rede.

Figura 3 - Arquitetura OpenFlow.



Fonte: Adaptado de (FOUNDATION, 2009).

Muitos fabricantes disponibilizam comutadores comerciais habilitados com o OpenFlow. Porém, também é possível encontrar comutadores virtualizados baseados em *software* com suporte ao OpenFlow; é o caso do comutador virtual Open vSwitch (OvS) (PFAFF et al., 2015), utilizado neste trabalho e descrito na Seção 1.3.3.

Até o presente momento, o protocolo OpenFlow conta com seis diferentes versões, sendo a versão 1.5 a mais recente. Porém, a versão 1.0 ainda é a mais utilizada, sendo usada inclusive neste trabalho. Cada uma das versões adiciona melhorias às funcionalidades do protocolo (LARA; KOLASANI; RAMAMURTHY, 2014). Segundo a especificação da versão 1.0 do OpenFlow (FOUNDATION, 2009), os comutadores OpenFlow utilizam a tabela de fluxo para encaminhar os pacotes, diferencialmente dos dispositivos de rede tradicionais que utilizam tabelas de encaminhamento e de repasse.

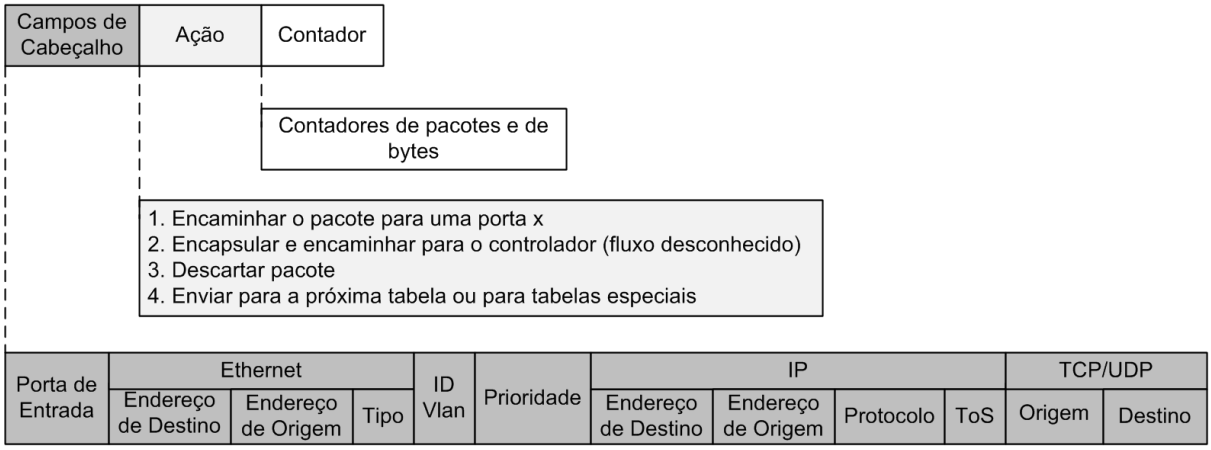
Uma tabela de fluxo consiste em um conjunto de entradas que são utilizadas pelo comutador para encaminhar os pacotes que compõem um fluxo de rede. Cada entrada na tabela de fluxo, como pode ser visto na Figura 4, possui três campos:

a) campos de cabeçalho: composto por múltiplos campos de cabeçalho que permitem estabelecer uma correspondência com o cabeçalho dos pacotes que são recebidos pelo comutadores (FOUNDATION, 2009);

b) ação: define como os pacotes do fluxo serão tratados; como exemplo de duas ações possíveis, um pacote pode ser encaminhado ou rejeitado;

c) contador: registra os números de bytes e pacotes dos fluxos que estabeleceram uma correspondência com uma entrada da tabela.

Figura 4 - Entrada da Tabela de Fluxo composta pelos campos de cabeçalho (regras), ação e contador.



Fonte: Versão de (COUTO, 2019) adaptada de (KREUTZ et al., 2015).

1.3.3 O Comutador Open vSwitch

No últimos anos, a virtualização mudou a maneira de se interagir com os sistemas computacionais. O número de máquinas virtuais (VM - *Virtual Machine*) superou o número de servidores físicos. Essa evolução não tardou para chegar até as redes de computadores. Na virtualização de rede, os comutadores virtuais são o principal provedor de serviços de rede para as VM. O OvS é um comutador virtual largamente utilizado por apresentar bom desempenho e permitir fácil programação de funções que o ambiente de rede virtualizado exige. Diferente dos dispositivos de rede tradicionais projetados para fornecer bom desempenho a partir de software ou hardware especializados, o OvS foi projetado para uso geral. Isso significa que ele pode ser executado em um *hardware* não especialista e também sobre as diversas distribuições de sistema operacional Linux existentes no mercado. Diferente de muitos *hardware switchs* que tiveram sua arquitetura modificada para trabalhar com o OpenFlow, o OvS já nasceu um comutador OpenFlow (PFAFF et al., 2015).

Na Figura 5, pode-se observar detalhes da arquitetura do OvS, que é composta por três componentes principais:

a) ovs-vswitchd: é o daemon[2] do OvS responsável por decidir qual ação o comutador irá executar. Estabelece comunicação direta com o controlador, por meio do protocolo OpenFlow, podendo receber configurações ou enviar dados de caráter informativo para o controlador;

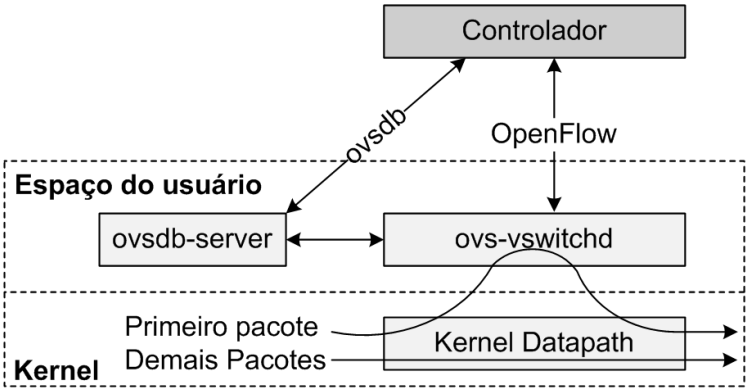
[2] Processo executado em segundo plano pelo sistema operacional do OvS.

b) *kernel datapath*: é um caminho que permite a comutação direta entre as portas do comutador. Possui um pequeno cache de memória responsável por carregar alguns registros da tabela de fluxo, o que permite que a verificação da correspondência de fluxo aconteça de maneira mais rápida. Dessa maneira, o comutador consegue encaminhar os pacotes do fluxo com maior velocidade;

c) ovsdb-server: consiste em um banco de dados que armazena as configurações do comutador.

A interação destes três componentes acontece da seguinte forma. Quando os pacotes de um fluxo chegam ao comutador OvS, o *kernel datapath* verifica se existe alguma instrução em sua memória cache do que fazer com os pacotes do fluxo. Caso seja a primeira vez que recebe o fluxo, provavelmente a sua memória cache ainda não terá instruções do que fazer com os pacotes do mesmo. Então, o primeiro pacote será enviado para o ovs-vswitchd, que irá consultar o banco de dados do ovsdb-server à procura de instruções do que fazer com os pacotes do fluxo recepcionado. Caso encontre, as instruções serão passadas para a memória cache do *kernel datapath* e os demais pacotes do fluxo serão processados e comutados diretamente para o seu destino. Entretanto, caso não encontre uma instrução do que fazer com os pacotes, o ovs-vswitchd se comunicará com o controlador utilizando o protocolo OpenFlow e solicitará instruções do que fazer com o fluxo. O controlador analisará a requisição e retornará instruções para o ovs-switchd. Então, o ovs-switchd armazenará as informações na base de dados do ovsdb-server e na memória cache do *kernel datapath*. Desta forma, o *kernel datapath* saberá o que fazer com os próximos pacotes do mesmo fluxo de dados (PFAFF et al., 2015).

Figura 5 - Arquitetura do comutador Open vSwitch.



Fonte: Adaptado de (PFAFF et al., 2015).

1.3.4 O Controlador OpenDaylight

Este estudo utiliza o controlador de SDN OpenDaylight por se tratar de um projeto aberto já bastante disseminado no meio acadêmico e industrial (KREUTZ et al., 2015; COX et al., 2017). Desenvolvido na linguagem de programação Java, ele possui uma aplicação nativa denominada *l2switch* composta pelo módulo *Loop Remover* responsável por gerar a árvore de cobertura da rede, similar à gerada pelo STP. A árvore de cobertura trata-se de um caminho padrão que abrange todos os nós da rede e visa eliminar caminhos redundantes que possam causar *loops* na rede. O ODL mantém um inventário da árvore de cobertura com o *status* do STP *forwarding* para os enlaces ativos e *discarding* para os enlaces inativos. Os *status* dos enlaces podem ser verificados por meio do console web de gerência do ODL (OPENDAYLIGHT, 2017).

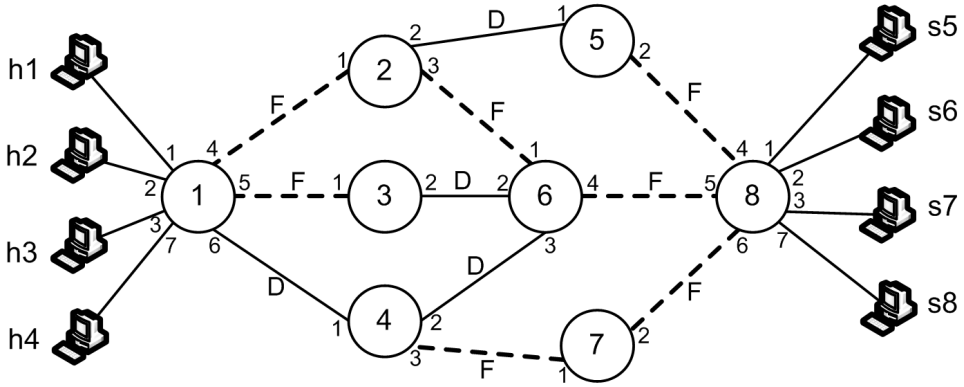
A utilização da aplicação *l2switch* permite que ODL opere em dois diferentes modos de funcionamento: proativo ou reativo.

1.3.4.1 Os Modos de Funcionamento Proativo e Reativo

No modo de funcionamento proativo (*proactive-flood mode*), o controlador ODL configura previamente regras de fluxos padrões nos comutadores da rede. Assim, os comutadores não precisam solicitar instruções aos controladores sobre o que fazer sempre que ingressarem novos fluxos (COSTA, 2016). Porém, como o comutador não possui uma relação de correspondência das suas portas de saída com endereço de destino, a regra de fluxo padrão replica o fluxo de dados que entra pela porta de entrada, denominada porta de ingresso, e o encaminha por todas as demais portas ativas, denominadas portas de egresso (OPENDAYLIGHT, 2017). Assim, utilizando a topologia apresentada na Figura 6 para exemplificar o comportamento descrito, assuma que “h1” deseja se comunicar com “s5”. Os enlaces tracejados sinalizam o caminho padrão determinado pela árvore de cobertura e identificados com a letra “F” de *forwarding*. Quando “h1” disparar o fluxo com destino a “s5”, este ingressará no comutador “1” pela porta “1” e será replicado e encaminhado pelas portas de egresso “2”, “3”, “4”, “5” e “7”, sinalizadas com o status *forwarding*. A única porta que não terá o fluxo replicado será a porta “6”, pois a mesma possui o *status discarding*. Agora, assuma também que “h2” deseja disparar um fluxo para “s6”. Quando “h2” disparar o fluxo com destino a “s6”, este ingressará no comutador “1” pela porta de ingresso “2” e será replicado para as portas de egresso “1”, “3”, “4”, “5” e “7”. Assim, os dois fluxos disputarão largura de banda em todos os caminhos ativos com *forwarding* e também nos enlaces conectados diretamente pelos *hosts*, o que irá prejudicar o desempenho dos fluxos.

Uma solução encontrada para evitar que os fluxos de ingresso sejam replicados pelas regras de fluxos padrões previamente configuradas na inicialização do ODL, consiste em inserir manualmente nova regra que corresponda às características do fluxo que será encaminhado (definido pelos campos do cabeçalho) tendo o campo de prioridade de execução definido com um valor mais alto do que o utilizado pelas regras padrões configuradas pelo ODL. Dessa maneira, o fluxo que corresponda a regra configurada manualmente não será tratado pelas regras padrões do ODL, já que o nível de prioridade destas é mais baixo (por padrão possuem valor igual a 2).

Figura 6 - Exemplo de topologia de rede na qual os enlaces tracejados indicam o caminho padrão designado pela árvore de cobertura gerada pelo ODL.



Já no modo de funcionamento reativo (*reactive mode*), quando um novo fluxo ingressa no comutador é realizada uma busca na tabela de fluxo por uma regra que corresponda ao novo fluxo. Se nenhuma regra for encontrada, uma amostra de um pacote pertencente ao fluxo (cabeçalho do pacote), ou até mesmo o pacote inteiro é enviado para o controlador. Quando o comutador tiver memória o suficiente para armazenar os pacotes que ingressam, o mesmo enviará ao controlador apenas o cabeçalho de um dos pacotes do fluxo (FOUNDATION, 2009; LARA; KOLASANI; RAMAMURTHY, 2014). Porém, se o comutador não tiver memória o suficiente para armazenar os pacotes que ingressam, o mesmo enviará o pacote completo ao controlador. Contudo, posteriormente o pacote será devolvido ao comutador de origem para que este o encaminhe ao seu destino. Esse tipo de pacote enviado para o controlador é denominado *packet-in* (FOUNDATION, 2009; LARA; KOLASANI; RAMAMURTHY, 2014). Então, após análise e processamento do *packet-in* pelo controlador, este enviará instruções com ações que serão inseridas na tabela de fluxo do comutador. A partir desse momento, o comutador saberá o que fazer com todos os fluxos que correspondam às entradas de fluxo recebidas (BARBECHO, 2017). As regras configuradas permanecerão nos comutadores até que o contador de tempo "*hard-timeout*" da regra expire (tempo padrão do *hard-timeout* é de 300 s) (OPENDAYLIGHT, 2017).

Caso o tempo expire a regra será excluída. Contudo, com a chegada de novos fluxos o processo de requisição de instruções ao controlador se repetirá. O modo reativo não realiza a replicação dos dados.

No modo reativo os comutadores trocam requisições e respostas *Address Resolution Protocol* (ARP) com os controladores. As interações envolvendo o protocolo ARP geram atrasos no encaminhamento dos fluxos. O uso do modo proativo com regras previamente configuradas pelo ODL e com regras específicas configuradas manualmente pelos administradores de rede diminuem o número de interações entre comutadores e controlador. Logo, o atraso no encaminhamento dos fluxos também diminui (OPENDAYLIGHT, 2017).

Em função das características dos modos de funcionamento do controlador, estes terão impacto direto na ocupação dos enlaces, consequentemente no balanceamento de carga. Desta maneira, optou-se por utilizar o modo de funcionamento proativo (padrão do ODL), nos experimentos da avaliação de desempenho detalhada no Capítulo 5, configurado com regras manuais responsáveis por encaminhar os fluxos de fundo pelo caminho padrão. A configuração manual das regras foi feita com o uso da ferramenta CURL, apresentada na Seção 5.1.3.

A próxima seção apresenta como é realizada a descoberta da topologia pelo controlador SDN.

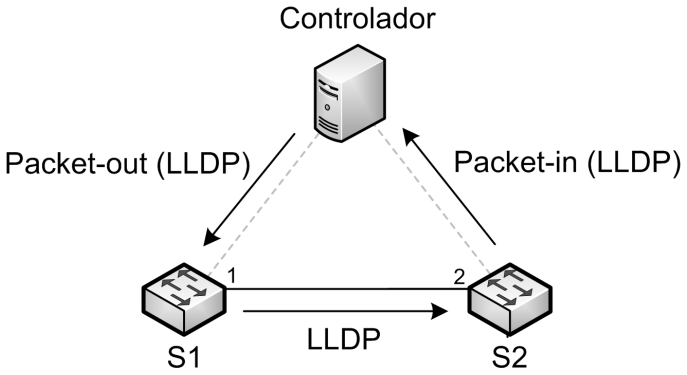
1.3.5 Descoberta da Topologia

O balanceamento de carga entre enlaces é feito realizando roteamento por multicaminhos. E um dos três componentes básicos do roteamento por multicaminhos é a computação de caminhos, apresentada no Capítulo 3. Mas para que esta ocorra é necessário que o algoritmo de computação de caminhos tenha uma visão global da topologia de rede e dos seus recursos disponíveis (SINGH; DAS; JUKAN, 2015). Contudo, uma das principais atribuições dos controladores de SDN envolve a descoberta da topologia de rede. Para tal, o controlador utiliza o protocolo OFDP (*OpenFlow Discovery Protocol*). Esse protocolo baseia-se no uso de um outro protocolo, o LLDP (*Link Layer Discovery Protocol*) que opera na camada de enlace do modelo OSI e permite que os dispositivos de rede anunciem informações aos seus dispositivos vizinhos como: nome do host, endereço MAC da interface, entre outras (AZZOUNI et al., 2017).

A Figura 7 apresenta um exemplo de funcionamento do OFDP. O controlador encapsula o pacote LLDP em um pacote *packet-out* e o envia para o comutador "s1". O pacote *packet-out* possui instruções para que "s1" encaminhe o pacote LLDP em todas as suas portas ativas, neste caso, para o comutador "s2". O comutador "s2" recebe o pacote LLDP, o encapsula em um pacote *packet-in* e o encaminha para o controlador. Assim, o controlador recebe o *packet-in* e passa a ter conhecimento que o comutador "s2" é vizinho

do comutador “s1”. Este processo é repetido com todos os comutadores da rede. Com as informações fornecidas pelo pacote LLDP, o controlador adquire uma visão global da topologia da rede.

Figura 7 - Processo de descoberta da topologia utilizando o protocolo LLDP.



Fonte: Adaptado de (AZZOUNI et al., 2017).

2 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos disponíveis na literatura que tratam de balanceamento de carga entre enlaces utilizando SDN, como também trabalhos que expõem o problema de encaminhamento de dados por múltiplos caminhos fazendo uso de soluções que substituem o uso do protocolo STP na rede Ethernet.

Em (BREDEL et al., 2014), os autores apresentam uma engenharia de tráfego logicamente centralizada com encaminhamento por multicaminhos que faz uso do controlador Floodlight e se baseia em caminhos disjuntos. Utilizam um algoritmo de seleção de caminhos que escolhe o caminho com menor número de fluxos mapeados. A quantidade de dados a serem transferidos deve ser informada ao algoritmo no início de sua execução. Assim, com base na taxa de transmissão do enlace é calculado um tempo médio para transmissão dos dados. Os novos fluxos são atribuídos aos enlaces com menores tempos de transmissão registrados. O trabalho ainda compara diferentes algoritmos de seleção de caminhos baseados em: *hash* dos campos do cabeçalho do pacote, escolha aleatória, *round-robin* ou quantidade de fluxos existentes no caminho. Os resultados apresentados mostram que o uso do algoritmo baseado na quantidade de fluxos apresenta um melhor desempenho em relação aos demais. Entretanto, os autores deixaram de incluir no trabalho a seleção por caminho baseada na taxa de tráfego e no volume de tráfego, geralmente utilizados para identificar a existência de congestionamento nos enlaces.

Em (RAMDHANI; HERTIANA; DIRGANTARA, 2016), os autores abordam a limitação do roteamento de caminho único imposto pelo uso do STP em redes Ethernet. Os autores apontam o uso do SDN como forma de ajudar no roteamento com multicaminhos. É proposto um balanceamento de carga com um controle de admissão de fluxo ativado sempre que a carga de dados no caminho atinge 80% da capacidade deste, o que ajuda a diminuir o congestionamento da rede. Os autores utilizam o algoritmo Depth-First Search (DFS) que realiza busca em profundidade para computar todos os caminhos não disjuntos entre origem e destino possíveis na rede. O modelo emprega estatísticas dos comutadores coletadas pelo controlador Ryu que ajudam na tomada de decisão. Uma avaliação de desempenho demonstra o bom desempenho do modelo proposto ao conseguir uma taxa de transferência mais estabilizada e latência mais baixa em relação ao roteamento de caminho único do STP.

(MALLIK; HEGDE, 2014) apresentam uma proposta de balanceamento de carga dinâmico utilizando multicaminhos com um mecanismo de controle do congestionamento que calcula o nível de carga de dados nos caminhos para saber se os mesmos podem ser utilizados na seleção de caminhos. A aplicação em conjunto com o controlador Floodlight procurar detectar se a carga do caminho está acima de um determinado limite e instrui os comutadores a encaminharem os fluxos por um caminho alternativo. Os autores apontam

como diferencial do trabalho a capacidade do seu modelo em identificar e reagir imediatamente ao desequilíbrio de carga e ao congestionamento de tráfego. Os autores relatam que são computados todos os possíveis caminhos entre a origem e o destino, caracterizando o uso de caminhos com enlaces não disjuntos. Contudo, é conhecido que esse tipo de computação de caminhos apresenta características de baixa tolerância a falhas.

(BHANDARKAR; KHAN, 2015) implementam um balanceamento de carga dinâmico que escolhe o caminho com maior largura de banda livre para rotear o tráfego. O trabalho apresenta uma avaliação de desempenho, com tráfego de dados gerado pela ferramenta Cbench, entre a solução implementada e o balanceamento de carga do controlador Floodlight baseado em *round-robin*. Segundo os autores, os resultados comprovam que com uso de sua solução em comparação com o balanceamento de carga *round-robin* foi possível encaminhar mais pacotes, como também diminuir a latência da rede. Contudo, o trabalho não apresenta o nível de congestionamento da rede, o que poderia ser feito mostrando a taxa de perda de pacotes do tráfego gerado.

Em (MUDIGONDA et al., 2010), os autores propõem o uso de um controlador central de rede composto por algoritmos codificados em *shell scripts* e *Perl scripts* responsáveis por computar os múltiplos caminhos com enlaces disjuntos na rede e converter cada um deles em uma árvore de cobertura distinta. Cada árvore de cobertura é mapeada como uma rede local virtual (VLAN - *Virtual Local Area Networks*). Para a descoberta da topologia é utilizado o protocolo LLDP. O *Smart Path Assignment in Networks* (SPAIN), como é denominado o projeto, utiliza simples *switchs* Ethernet de uso comercial (COTS - *commercial off-the-shelf*). Contudo, a implementação do SPAIN requer que sejam realizadas modificações no *kernel* do *host* final. Tais modificações são necessárias para que o pacote transmitido pelo *host* receba uma identificação que permita seu acesso aos caminhos virtuais mapeados pelo controlador. O trabalho mostra que o SPAIN foi capaz de aumentar a largura de banda agregada em redes simuladas e experimentais para centros de dados. Porém, os autores acreditam que o desempenho do SPAIN em redes arbitrárias possa ser tão bom quanto em redes com topologia regular, como por exemplo em redes FatTree e BCube, utilizadas em *datacenters*). Um ponto negativo ao uso do SPAIN está no fato de ser necessário a modificação no *kernel* dos *hosts* da rede.

Em (PERLMAN; TOUCH, 2009) os autores propõem o uso de um protocolo denominado *TRansparent Interconnection of Lots of Links* (TRILL) com atuação direta na camada de enlace. A utilização do TRILL visa substituir o uso do protocolo STP em redes Ethernet e proporcionar encaminhamento de pacotes com a possibilidade de utilização de múltiplos caminhos. Seu funcionamento conta com o uso de um dispositivo de rede denominado *Routing Bridges* (RBridges) que substitui o comutador de rede tradicional. A RBridges possui características de dispositivos de camada de rede e camada de enlace. Cada RBridge recebe um ID de identificação utilizado no encaminhamento dos pacotes entres as RBridges.

Para que o pacote Ethernet deve ser encapsulado pelo protocolo TRILL para que desta maneira possa ser encaminhado pela rede virtual estabelecida entre as RBridges. Seu cabeçalho conta com os seguintes campos de identificação: RBridge de origem, da RBridge de destino e de *hop count* que guarda a contagem de saltos na rede TRILL (COSTA et al., 2012). Os caminhos são mapeados por meio da utilização do protocolo de roteamento IS-IS executado diretamente na camada de enlace. Contudo, o TRILL não leva em consideração a condição do tráfego no caminho que será utilizado, a seleção do caminho baseia-se na escolha do que apresentar menor número de saltos.

Em (NGUYEN; KIM, 2015) os autores propõem um novo método de roteamento por multicaminhos denominado *Accumulative-Load Aware Routing* (ALAR), baseado em acúmulo de carga nos enlaces utilizando SDN. Tal método considera a carga dos fluxos acumulada em cada enlace pelos contadores do controlador. O ALAR realiza a computação de caminhos baseada no algoritmo de Dijkstra e armazena tal informação em uma base de dados juntamente com os respectivos custos encontrados a partir do volume de dados registrado nos enlaces de cada caminho. O artigo ainda propõe uma comparação de desempenho do ALAR com o roteamento convencional de caminho mais curto utilizando o protocolo OSPF. Os experimentos são realizados no emulador de rede mininet e o no controlador de rede OpenIRIS. Os resultados finais concluem que o ALAR consegue diminuir o atraso de ponta a ponta, enquanto proporciona o aumento da taxa de vazão. Contudo, diferentemente deste estudo, o ALAR utiliza caminhos não disjuntos e custo para escolha dos caminhos baseado no volume de dados aferido em cada caminho.

Em (BANFI et al., 2016) apresentam uma solução de encaminhamento de dados utilizando múltiplos caminhos baseado em uma arquitetura de SDN. Além disso, o trabalho também trata do problema de reordenamento de pacotes no lado receptor, pois o encaminhamento por multicaminhos tende a gerar desordem na sequência de envio dos pacotes. O MultiPath Software-Defined Networks (MPSDN), como os autores denominam seu algoritmo, utiliza o controlador de SDN Ryu e comutadores OpenvSwitch em uma rede emulada pelo mininet. A descoberta da topologia é realizada utilizando o protocolo LLDP e a computação de caminhos utiliza a teoria do fluxo máximo com emprego do algoritmo de Edmonds-Karp para encontrar os caminhos. A seleção do caminho é feita por um escalonador *Weighted Round-Robin* (WRR). E para evitar o impacto do reordenamento de pacotes no lado receptor, os autores introduziram um *buffer* de sequenciamento no comutador de borda do lado receptor. O *buffer* é responsável por armazenar temporariamente os pacotes recebidos antecipadamente, para posterior liberação em conjunto com os pacotes retardatários identificados pelo número de sequência. O trabalho realiza ainda uma avaliação de desempenho entre uma implementação do *Multipath* TCP (MPTCP) e do MPSDN, na qual a aplicação proposta apresenta resultados semelhantes ao uso do MPTCP. Porém, com o diferencial de não precisar alterar o kernel do *host* conforme é feito ao se utilizar o MPTCP.

Este trabalho, de forma similar às propostas de (RAMDHANI; HERTIANA; DIRGANTARA, 2016; MALLIK; HEGDE, 2014), também propõe um mecanismo de balanceamento de carga baseada em SDN, que conta com um controle de comutação entre enlaces ativado sempre que o nível de carga no caminho atingir um determinado limiar. Além disso, esse mecanismo ainda verifica também se o novo caminho possui maior largura de banda livre em relação à do caminho atual. Assim, de um modo geral, ele evita que sejam realizadas comutações de fluxos para caminhos com níveis de ocupação saturados e que possam causar congestionamento do novo caminho escolhido. Também de forma similar à proposta apresentada por (BANFI et al., 2016), este trabalho utiliza a teoria de fluxo máximo com emprego do algoritmo de Edmonds-Karp para encontrar caminhos com enlaces disjuntos na rede.

3 ROTEAMENTO MULTICAMINHOS

O uso de mais de um caminho para a comunicação entre origem e destino emergiu com a crescente demanda de serviços ofertados na Internet. Sites que ofertam serviços de imagens e fotografias, redes sociais, *streaming* de vídeos e vídeo sob demanda (VoD, do inglês *Video-on-Demand*) constituem um forte incentivo para o ingresso de novos usuários à Internet e juntos com eles todos os seus dispositivos pessoais que possuem conectividade de rede, os famosos *gadgets*[3] (SINGH; DAS; JUKAN, 2015). O número desses sites vem crescendo ao longo dos anos. De forma a ter uma ideia desse crescimento, a Tabela 1 (Internet Systems Consortium, 2019) apresenta o número de *hosts* conectados à Internet entre 2001 e 2018.

Esse novo conjunto de serviços contribui para o crescente volume de dados nas redes de computadores. Um estudo realizado pela *International Data Corporation* (IDC) e patrocinado pela Seagate em 2018 prevê que o volume de dados na Internet no ano de 2025 chegará a 175 Zettabytes (GANTZ; REINSEL; RYDNING, 2018). A manipulação de grandes volumes de dados pode implicar congestionamento na rede que aparecerá para os usuários na forma de atrasos de ponta a ponta e baixas taxas de transferências de dados.

Tabela 1 - Número de hosts conectados à Internet nas últimas duas décadas.

Ano	**Hosts**	**Ano**	**Hosts**	**Ano**	**Hosts**
2001	109.574.429	2007	433.193.199	2013	963.518.598
2002	147.344.723	2008	541.677.360	2014	1.010.251.829
2003	171.638.297	2009	625.226.456	2015	1.012.706.608
2004	233.101.481	2010	732.740.444	2016	1.048.766.623
2005	317.646.084	2011	818.374.269	2017	1.062.660.523
2006	394.991.609	2012	888.239.420	2018	1.003.604.363

O uso de multicaminhos pode ajudar a balancear a carga de dados entre os caminhos alternativos, de modo a diminuir o atraso e a elevar as taxas de transferências (SINGH; DAS; JUKAN, 2015).

Em (TSAI; MOORS, 2006; SINGH; DAS; JUKAN, 2015), são apresentadas algumas vantagens de se utilizar roteamento multicaminhos:

a) tolerância a falhas: o uso de caminhos alternativos reduz a probabilidade

[3] Termo tecnológico para designar dispositivos eletrônicos portáteis (smartphones, smartwatchs, tablets, etc.).

de interrupção total na comunicação caso um caminho apresente falha. Como o tráfego é divido pelo caminhos existentes, em caso de falha serão perdidos apenas alguns pacotes;

b) balanceamento de carga: o tráfego pode ser distribuído pelos caminhos existentes de forma a não sobrecarregar um único caminho;

c) agregação de banda: a distribuição dos dados em vários fluxos por caminhos diferentes aumenta a largura de banda efetiva. Assim, é possível utilizar uma largura de banda maior do que quando utilizado um único caminho;

d) segurança: o uso de vários caminhos para distribuir a informação dificulta ataques de interceptação completa dos dados por um atacante.

A técnica de roteamento por multicaminhos explora os recursos físicos da rede utilizando vários caminhos entre a origem e o destino para escoar o tráfego de dados (TSAI; MOORS, 2006).

Um exemplo de protocolo de roteamento que suporta multicaminhos é o *Open Shortest Path First* (OSPF). Esse protocolo permite o roteamento por multicaminhos desde que os caminhos possuam o mesmo custo. O custo atribuído às interfaces dos roteadores não leva em consideração enlaces congestionados; ou seja, o custo do OSPF é estático, podendo ser modificado apenas pelo administrador da rede (BURIOL, 2003). Porém, existe uma extensão para o OSPF denominada *Traffic Engineering* (TE) que fornece informações do estado dos enlaces que podem ser utilizadas para determinar as melhores rotas de acesso da rede. Tal extensão distribui informações de desempenho dos enlaces da rede incluindo atraso de propagação do enlace, variação de atraso, perda de enlace, largura de banda residual, largura de banda disponível e largura de banda utilizada (LONG et al., 2018).

O roteamento de multicaminhos pode ser dividido em três componentes básicos: a computação de caminhos, a divisão de tráfego e a seleção de caminhos (SINGH; DAS; JUKAN, 2015; PRABHAVAT et al., 2012).

A computação de caminhos tem por objetivo encontrar todos os caminhos existentes entre uma origem e um destino. Para que o processo de busca seja eficaz, o algoritmo responsável pela computação de caminhos precisa ter conhecimento global da topologia da rede. Após a descoberta da topologia, o algoritmo precisa identificar os caminhos da origem até o destino com base em três cenários (SINGH; DAS; JUKAN, 2015):

a) nós disjuntos: o caminho é composto por nós (que não sejam a origem e o destino) não compartilhados por outros caminhos. O fato de não compartilhar nós significa que também não haverá compartilhamento de

enlaces, exceto para enlaces *broadcast* e redes *non-broadcast multi-access* (NBMA). Esta configuração proporciona maior tolerância a falhas, já que os caminhos são totalmente independentes. Contudo implica maior gasto com infraestrutura, pois para se obter mais caminhos é necessário criar mais nós e enlaces. Esse cenário é exemplificado na Figura 8(a), que apresenta um grafo composto de 10 nós e 14 enlaces, sendo o nó "A" a origem e o nó "J" o destino de dois fluxos que percorrem caminhos totalmente independentes e livres do compartilhamento de nós e enlaces;

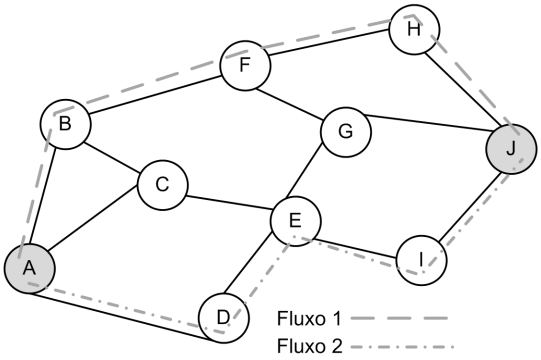
b) enlaces disjuntos: o caminho é composto por nós que podem ser compartilhados por outros caminhos, porém os enlaces não são compartilhados. Este cenário possui menor tolerância a falhas em relação ao cenário anterior. Isso porque, ao ocorrer falha em um nó, todos os caminhos que fizerem uso deste serão afetados também. O mesmo não acontecerá se a falha ocorrer em um enlace ao invés de em um nó. O uso de enlaces disjuntos pode ajudar a aumentar a quantidade de largura de banda total e a diminuir o congestionamento na rede (SUN et al., 2012). Um exemplo desse cenário é apresentado na Figura 8(b), onde existem três fluxos percorrendo caminhos que não compartilham enlaces, mas apresentam compartilhamento do nó "E" pelos fluxos 2 e 3;

c) enlaces não disjuntos: exemplificado na Figura 8(c), tanto os nós quanto os enlaces de um caminho podem ser compartilhados pelos demais caminhos da rede. A falta de restrição quanto ao uso de nós e enlaces comuns torna mais fácil a computação dos caminhos (SINGH; DAS; JUKAN, 2015). Contudo este cenário é considerado o pior caso entre os três, pois caso ocorra uma falha em um nó ou enlace, essa falha afetará todos os caminhos que compartilharem tais recursos físicos.

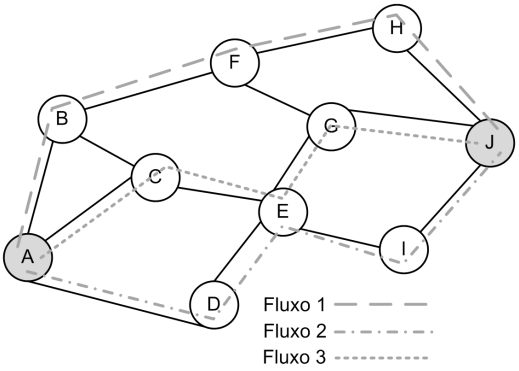
O desempenho do algoritmo na computação de caminhos depende do número de nós e enlaces existentes na topologia. Segundo (SINGH; DAS; JUKAN, 2015), estabelecer um número ideal de caminhos pode reduzir a complexidade do processamento empregado. Além disso, caminhos não disjuntos devem ser evitados, já que o compartilhamento de nós e enlaces significa ter que lidar com baixa tolerância a falhas. Outra questão é quanto à largura de banda, pois caminhos com enlaces compartilhados também terão a sua largura de banda compartilhada.

Os mecanismos de balanceamento de carga utilizados neste trabalho fazem uso de caminhos não disjuntos no caso do mecanismo de Seth e caminhos com enlaces disjuntos no caso do mecanismo proposto denominado MLB. A Seção 3.1 apresenta o teorema de fluxo máximo e corte mínimo utilizado pelo mecanismo MLB para computar caminhos com enlaces disjuntos.

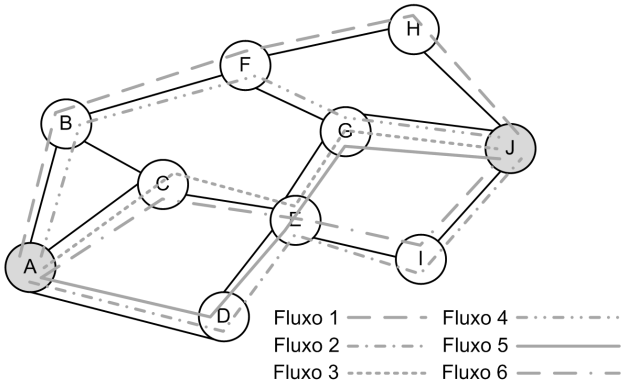
Figura 8 - Exemplos de cenários de computação de caminhos considerando o nó “A” como origem e “J” como destino.



(a) Nós disjuntos



(b) Enlaces disjuntos



(c) Enlaces não disjuntos

Seguida à computação de caminhos, deve ser realizada a divisão do tráfego destinado aos caminhos. Segundo (PRABHAVAT et al., 2012), a divisão de tráfego apresentada na Figura 9 pode ser classificada em:

a) nível de pacote: o tráfego é dividido na menor escala possível, ou seja, em pacotes. Cada pacote é tratado individualmente. Ou seja, os pacotes podem ser transmitidos cada um por um caminho, o que pode gerar problemas de ordenação de pacotes no destino final.

b) nível de fluxo: são estabelecidos identificadores para o pacote com base nos campos do seu cabeçalho e todos os pacotes que se dirigirem para o mesmo destino são agrupados juntamente;

c) nível de subfluxo: um fluxo de pacote pode ser dividido em fluxos menores, os subfluxos. Ou seja, um subconjunto de pacotes de um fluxo original. A divisão é feita dependendo do protocolo empregado. Um exemplo de protocolo que utiliza subfluxos é o *MultiPath* TCP (MPTCP). Neste, os subfluxos são montados por um escalonador existente na camada de

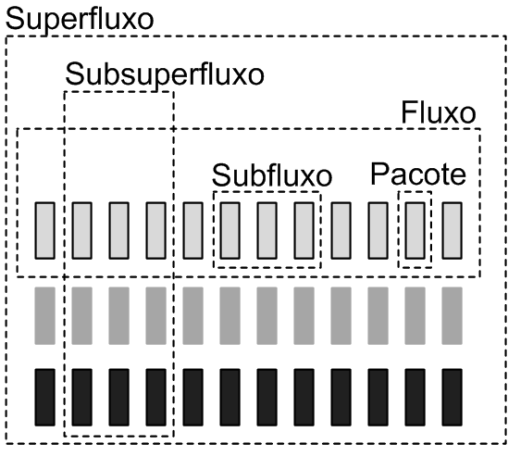
transporte. Cada subfluxo recebe um identificador para que o *host* de destino possa identificar e ordenar os subfluxos que receba (SCHARF; FORD, 2013);

d) nível de superfluxo: ocorre quando utilizado uma função hash para gerar identificadores do fluxo com base nos campos do cabeçalho dos pacotes que compõem tal fluxo. Pode ocorrer que o resultado calculado pela função *hash* seja igual para vários fluxos, mesmo que os cabeçalhos de seus pacotes sejam distintos. Então, diferentes fluxos que possuam o mesmo identificador retornado pela função *hash* são agrupados formando o superfluxo.

e) nível de sub-superfluxo: trata-se de um subconjunto de superfluxos que satisfazem uma determinada condição de divisão, semelhante ao que ocorre com o subfluxo.

De maneira a evitar problemas com o ordenamento de pacotes que chegam ao destino, gerados pela troca de caminhos durante o balanceamento de carga, optou-se por utilizar a divisão de tráfego no nível de fluxo baseada nos seguintes identificadores de pacotes: IP de origem, IP de destino, protocolo e porta de destino. Assim, os pacotes de um mesmo fluxo são encaminhados pelo mesmo caminho configurado pelo balanceador de carga. Dessa forma, a entrega de pacotes fora de ordem acontece com menos frequência, o que ajuda a diminuir a perda de pacotes e o atraso na entrega.

Figura 9 - Categorias da divisão de tráfego.



Fonte: Adaptado de (PRABHAVAT et al., 2012)

Já a seleção de caminhos pode ser classificada em função do tipo de seletor utilizado (PRABHAVAT et al., 2012):

a) *round robin*: o tráfego é encaminhado por todos os caminhos computados obedecendo uma sequência cíclica;

b) informação do pacote: o tráfego é encaminhado com base em informações contidas no cabeçalho do pacote;

c) condição de tráfego: pode considerar a vazão do tráfego, o volume do tráfego ou o número de fluxos ativos no caminho;

d) condição da rede: pode considerar o tamanho da fila, o atraso, o *jitter* ou a perda de pacotes no caminho.

As implementações de balanceamento de carga utilizadas neste trabalho se basearam na seleção de caminho por condição de tráfego, pois utilizam a vazão do tráfego no caminho (mecanismo MLB) e o volume de tráfego no caminho (mecanismo de Seth) como medidas de custo para identificar o melhor caminho.

3.1 Teorema do Fluxo Máximo e Corte Mínimo

Esta seção apresenta o teorema de fluxo máximo e corte mínimo e como são implementados pelo algoritmo de Edmonds e Karp, utilizado pelo mecanismo de balanceamento de carga MLB para computar caminhos com enlaces disjuntos em uma rede de computadores.

Segundo (CORMEN et al., 2012), do mesmo modo que é possível modelar um mapa rodoviário como um grafo dirigido para encontrar o caminho entre dois pontos, também é possível interpretar um grafo dirigido como sendo uma "rede de fluxo" utilizada para modelar problemas envolvendo: tráfego veicular, logística de entrega de materiais, distribuição de líquidos por tubulações, distribuição de correntes elétricas por condutores, fluxos de dados em redes de computadores, entre outros.

O cálculo do fluxo máximo serve para se encontrar o número máximo de fluxos que poderão percorrer a rede de fluxos saindo de uma fonte para um sorvedouro; como também pode ser utilizado para encontrar os caminhos com enlaces disjuntos de um grafo (CORMEN et al., 2012).

Do ponto de vista da teoria dos grafos, uma rede de fluxo pode ser representada por um grafo dirigido $G = (V, E)$ no qual,

$$V = \{\textit{conjunto de nós da rede}\} \textit{ e } E = \{\textit{conjunto de arestas da rede}\}.$$

Cada aresta tem uma capacidade não negativa e pode ser representada por (u, v) no qual, $u = \textit{vértice de origem}$ e $v = \textit{vértice de destino}$. Assim, o conjunto de arestas é definido

por $(u, v) \in E$ e sua capacidade é definida por $c(u, v) \geq 0$. Além disso, se $(u, v) \notin E$, então pode-se definir que $c(u, v) = 0$. Distingue-se o nó fonte como "s" e o nó sorvedouro como "t".

Um fluxo f em G deve satisfazer as seguintes propriedades:

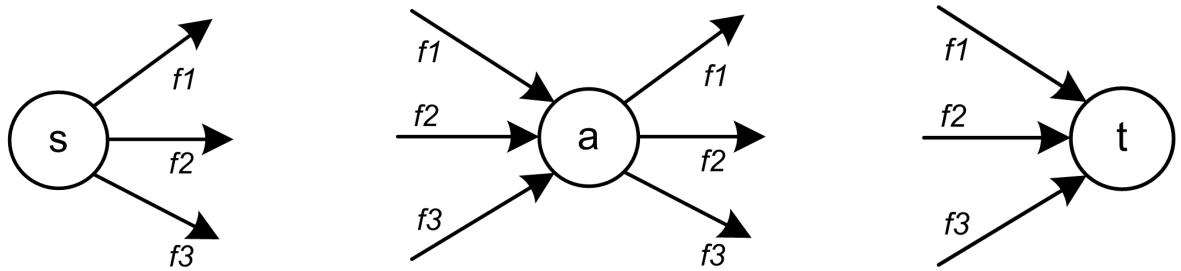
a) Restrição de capacidade: Para todo $u, v \in V$, exige-se $0 \leq f(u, v) \leq c(u, v)$;

b) Conservação do fluxo do nó fonte (que sai): Para "s", impõe-se $\sum_s f(u, v) \leq c(u, v)$. A Figura 10(a) apresenta o comportamento dos fluxos em um nó fonte segundo a propriedade de conservação do fluxo do nó fonte;

c) Conservação de fluxo do nó intermediário: Para todo $u \in V - \{s, t\}$, ou seja, todos os nós do conjunto V, exceto os nós de origem "s" e destino "t", impõe-se $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$, onde o fluxo de saída do nó tem que igual ao fluxo de entrada. A Figura 10(b) apresenta o comportamento dos fluxos em um nó intermediário segundo a propriedade de conservação do fluxo em nós intermediários;

d) Conservação do fluxo do nó de destino (que chega): Para "t", impõe-se $\sum_t f(v, u) \leq c(u, v)$. A Figura 10(c) apresenta o comportamento dos fluxos em um nó de destino segundo a propriedade de conservação do fluxo do nó de destino.

Figura 10 - Propriedades de conservação de fluxos.



(a) conservação do fluxo do nó fonte.

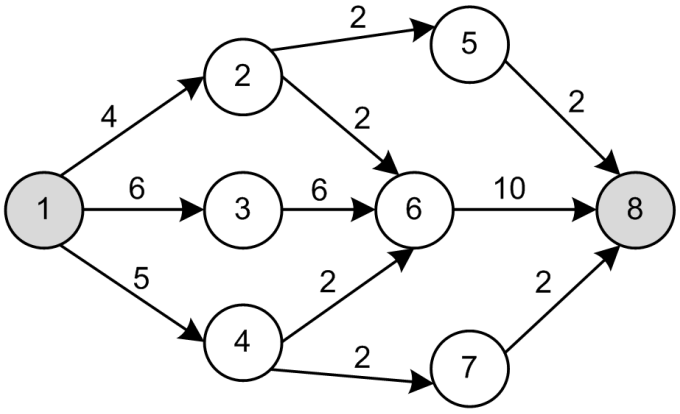
(b) conservação do fluxo em nós intermediários.

(c) conservação do fluxo do nó sorvedouro.

No problema do fluxo máximo temos uma rede de fluxo G com fonte "s" e sorvedouro "t", e deseja-se encontrar o fluxo máximo entre eles. A Figura 11 apresenta um grafo dirigido com o nó fonte representado pelo vértice "1", o nó de destino pelo vértice "8" e a capacidade suportada por cada aresta. Os problemas de fluxo máximo

podem ser modelados como problemas de programação linear e resolvidos por meio do método Simplex. Porém, algoritmos de "caminho aumentado" costumam ser mais eficientes na resolução dos problemas de fluxo máximo (HILLIER; LIEBERMAN, 2006). Para resolver o problema do fluxo máximo, pode-se utilizar o método de Ford-Fulkerson, baseado em caminho aumentado, apresentado na próxima seção.

Figura 11 - Grafo dirigido, apresenta a capacidade de cada aresta. Nó fonte "s" representado pelo vértice "1" e o nó sorvedouro "t" representado pelo vértice "8".



3.1.1 O Método de Ford-Fulkerson

Esta seção apresenta o funcionamento do método de Ford-Fulkerson, implementado pelo algoritmo de Edmonds e Karp (apresentado na próxima seção) para encontrar o fluxo máximo de uma rede.

A execução do método de Ford-Fulkerson se baseia em três importantes conceitos:

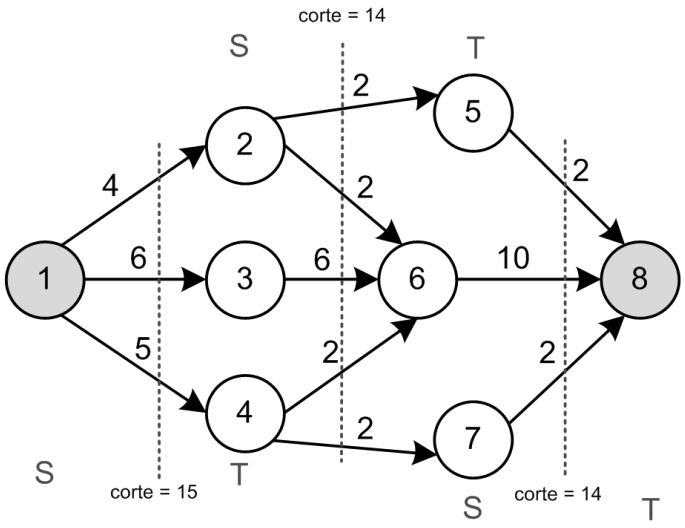
a) rede residual: consiste em uma rede de fluxo G_f formada a partir da rede original G com um fluxo f e que tem por função mostrar a "capacidade residual" e os fluxos adicionais submetidos a mesma. A rede residual G_f é representada por arestas sinalizadas com os valores de sua capacidade e dos fluxos existentes. Cada aresta de G_f pode receber um fluxo adicional menor ou igual a sua "capacidade residual". Assim, a "capacidade residual" pode ser representada por $c_f(u, v) = c(u, v) - f(u, v)$;

b) caminho aumentador: trata-se do caminho "p" do nó fonte "s" até o nó sorvedouro "t", na rede residual G_f. A quantidade máxima de aumento de fluxo em um caminho aumentador "p" é denominada de "capacidade

residual de p", expressa por "$c_f = min c_f(u, v) : (u, v)$ *está em* p". Isso significa que a capacidade residual do caminho "c_f" receberá o menor valor registrado da capacidade residual de cada enlace"$c_f(u, v)$" existente no caminho p. O fluxo f é aumentado gradualmente enquanto o valor da "c_f" se mantiver maior que "0";

c) cortes de rede: um corte (S, T) de uma rede de fluxo $G = (V, E)$ é uma partição de V em S e $V - S$ em T, tal que $s \in S$ e $t \in T$. Ou seja, S é o conjunto de nós de origem e T é o conjunto dos nós de destino entre cada corte mínimo realizado na rede. Um corte é dito como "corte mínimo" da rede quando sua capacidade é mínima em relação a todos os outros cortes da rede, como apresentado na Figura 12. Conforme apresentado em (CORMEN et al., 2012), o fluxo máximo em uma rede é limitado por cima pela capacidade de um corte mínimo da rede.

Figura 12 - O fluxo máximo da rede é dado em função do corte que apresentar o menor valor de somatório das capacidades dos enlaces, neste caso 14.



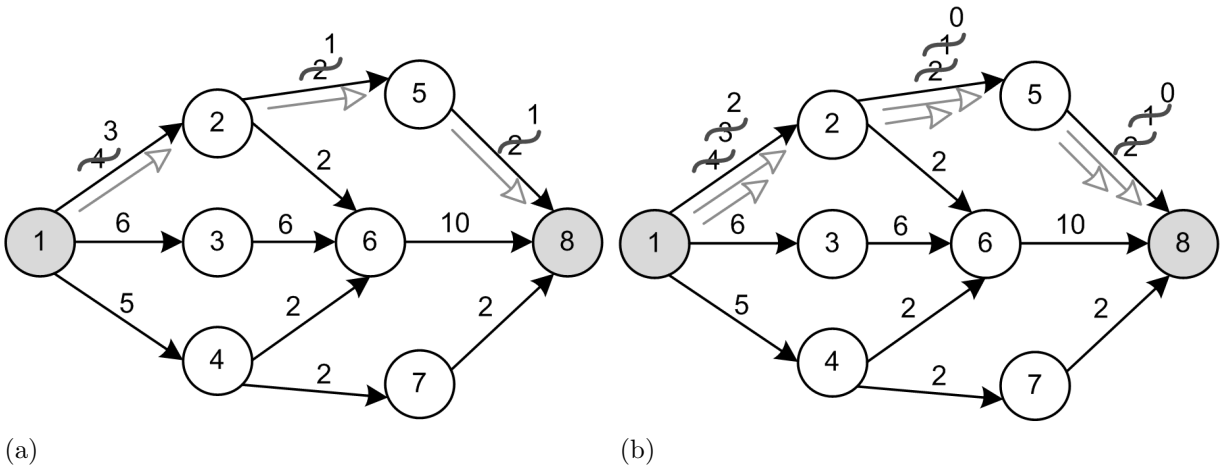
A Figura 13 apresenta um exemplo de rede residual. O método realiza o aumento do valor do fluxo, que começa em $f(u, v) = 0$ para todo $u, v \in V$ e progride até alcançar o valor de fluxo máximo para um caminho. Cada incremento de fluxo no caminho aumentador é representado na rede residual G_f. A ideia é repetir esse processo até que não existam mais caminhos aumentadores na rede residual G_f.

Na Figura 13(a) é possível observar que com um fluxo f enviado pelo caminho formado pelos nós "1-2-5-8", a capacidade dos enlaces é decrementada em um fluxo. De acordo com o conceito apresentado de caminho aumentador, ainda é possível transmitir um

fluxo por este caminho, já que a capacidade residual do caminho c_f ainda apresenta valor "1". Com a transmissão de um segundo fluxo f a capacidade dos enlaces é decrementada novamente em um fluxo, conforme apresentado a Figura 13(b). Com a capacidade residual do caminho c_f apresentando o valor "0" não é possível mais transmitir novos fluxos por este caminho. O enlace formado pelos nós "1-2" é o único enlace do caminho 1-2-5-8 ainda com capacidade residual para transmitir até dois fluxos. Desta forma, o caminho formado pelos nós "1-2-6-8" ainda pode ser utilizado por até dois fluxos. Este processo de envio de fluxo com a subtração da capacidade residual deve ser repetido enquanto existirem caminhos com capacidade residual $c(u, v) \geq 1$. Quando todos os caminhos apresentarem sua $c(u, v) = 0$, significará que o fluxo máximo da rede é igual ao número de fluxos enviados.

Chegado a este ponto, pode-se chamar atenção para o seguinte fato, se a capacidade de todos os enlaces da rede de fluxo for configurada com valor igual a "1", será possível estabelecer uma relação de uso dos caminhos por apenas um fluxo. Além disso, o caminho que tiver um enlace compartilhado com um outro caminho e a capacidade deste apresentar-se zerada, não terá fluxo atribuído ao mesmo. Pois, isso significa que o enlace já está em uso por um outro caminho. Por meio deste processo pode-se encontrar os caminhos com enlaces disjuntos de uma rede de fluxo. A Seção 3.1.3 apresenta uma explicação para este processo de maneira mais detalhada.

Figura 13 - Rede residual G_f formada por caminhos aumentadores e suas capacidades entre o nó "1" e o nó "8".



(a) (b)

Legenda: (a) Capacidade residual decrementada pelo primeiro fluxo representado pela seta de ponta aberta.
(b) Capacidade residual decrementada pelo segundo fluxo representado pela seta de ponta aberta.

A próxima seção apresenta uma implementação do método de Ford-Fulkerson denominado algoritmo de Edmonds e Karp.

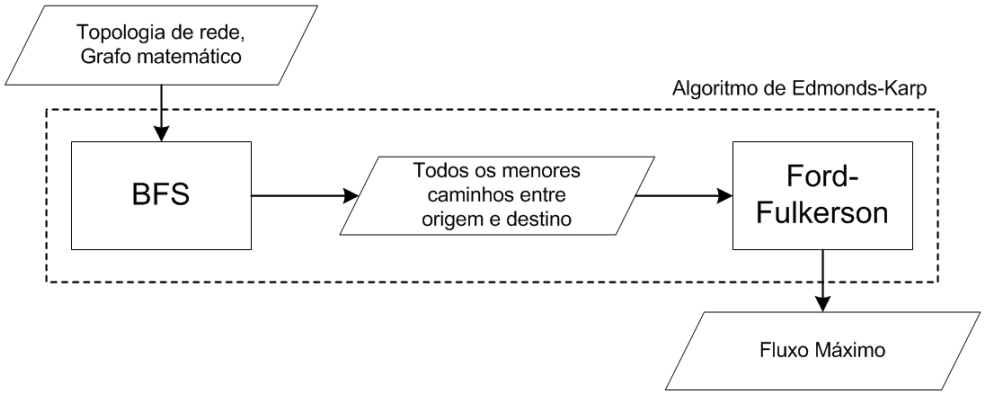
3.1.2 Algoritmo de Edmonds e Karp

Um ponto importante que deve ser levado em consideração é quanto ao tempo de execução do método de Ford-Fulkerson, que depende de como são encontrados os caminhos aumentadores. A busca pelos caminhos aumentadores é de extrema importância para o bom desempenho do algoritmo. Dependendo da implementação do algoritmo, o tempo de execução pode apresentar baixa eficiência e chegando até mesmo, em alguns casos, a não terminar a sua execução (CORMEN et al., 2012). Uma solução viável é utilizar a busca em largura[4] para realizar a computação de caminhos.

Segundo (CORMEN et al., 2012), o método de Ford-Fulkerson quando implementado com o algoritmo de busca em largura Breadth First Search (BFS) recebe a denominação de algoritmo de Edmonds e Karp.

A Figura 14 apresenta um diagrama que mostra o funcionamento do algoritmo de Edmonds e Karp. Neste, uma topologia de rede transformada em uma estrutura de grafo, com vértices definidos como origem e destino, é submetida como parâmetro de entrada ao algoritmo BFS. O BFS então realiza o processamento dos dados e retorna todos os menores caminhos encontrados entre origem e destino e os submete a implementação do algoritmo do método de Ford-Fulkerson, responsável por retornar o valor do fluxo máximo da rede.

Figura 14 - Diagrama de funcionamento do algoritmo de Edmonds e Karp.



- Implementação do algoritmo BFS

  A Figura 15 apresenta um pseudocódigo do funcionamento do Breadth First Search (BFS). A estrutura de repetição na linha 2 inicializa todos os vértices do grafo,

[4] Busca em um grafo que começa pelo vértice raiz e explora todos os vértices vizinhos. Em seguida, para cada um dos vértices explorados, se exploram os seus vértices vizinhos e assim sucessivamente até não existirem mais vértices inexplorados.

exceto o vértice de origem “s”, com o valor “BRANCO”, com distância (d) “infinita” e predecessor (π) nulo (Null), pois estes vértices ainda não foram visitados. As linhas 7-9 inicializam o vértice origem “s” com “CINZA”, pois considera que ele encontra-se visitado ao iniciar a busca. A linha 10 inicializa a estrutura de fila “Q” vazia, com 0 elementos. A linha 11 adiciona o vértice “s” à estrutura de fila “Q”. Na linha 12, o bloco “enquanto” dá início à visitação dos vértices. O processo de visitação é repetido até que a fila “Q” fique vazia. Na linha 13, o primeiro elemento da fila é movido para “u”. Na linha 14 é considerado cada vértice “v” que seja adjacente ao vértice “u”. Se o vértice adjacente “v” ainda não tiver sido visitado, então este é associado a “CINZA”, “d” recebe a distância de “u” + 1 e π recebe o vértice “u”. Na linha 19 a estrutura da fila “Q” recebe o vértice visitado. Quando todos os vértices “v”s adjacentes ao vértice “u” tiverem sido visitados, “u” receberá “PRETO”, como mostrado na linha 22. Então, o processo é repetido novamente até que todos os vértices e seus adjacentes sejam visitados.

Figura 15 - Pseudocódigo do algoritmo BFS que realiza busca em largura.

BFS - Breadth-First Search

```
1  BFS (G, s)
2      para cada vértice u ∈ V[G]-{s}  faça
3          u.cor = BRANCO
4          u.d = ∞
5          u.π = Null
6      fim
7      s.cor = CINZA
8      s.d = 0
9      s.π = Null
10     Q = 0
11     E_NQUEUE(Q, s)
12     enquanto Q ≠ 0  faça
13         u = D_EQUEUE(Q)
14         para cada v = Adj[u]  faça
15             se v.cor == BRANCO  então
16                 v.cor = CINZA
17                 v.d = u.d + 1
18                 v.π = u
19                 E_NQUEUE(Q, v)
20             fim
21         fim
22         u.cor = PRETO
23     fim
24 fim
```

Fonte: adaptado de CORMEN et al., 2012.

- Implementação do algoritmo para o método Ford-Fulkerson

  Para cada iteração do método, descobrem-se os caminhos aumentadores "p" que serão utilizados para encontrar o fluxo máximo por meio do incremento gradual do fluxo f. No pseudocódigo apresentado na Figura 16, as linhas 2-3 inicializam o fluxo f como 0. A estrutura de repetição "enquanto" da linha 5 mantém a execução do laço enquanto existirem caminhos aumentadores de "s" para "t" na rede residual G_f. A linha 6 atribui a $c_f(p)$ a menor capacidade encontrada entre as arestas (u, v) do caminho "p". Na linha 7 é iniciado um laço que repete sua execução para cada aresta (u, v) do caminho "p". A estrutura condicional da linha 8 testa se a aresta residual é (u, v). Em caso positivo, ao fluxo f é somado $c_f(p)$. No caso contrário, o fluxo f diminui de $c_f(p)$. Quando acabam os caminhos aumentadores, o fluxo f é considerado o fluxo máximo.

Figura 16 - Pseudocódigo do método de Ford-Fulkerson.

FORD E FULKERSON

```
1  Ford-Fulkerson (G, s, t)
2      para cada aresta (u, v) ∈ G.E faça
3          f(u, v) = 0
4      fim
5      enquanto existir um caminho p de s a t na rede residual G_f  faça
6          c_f(p) = min{c_f(u, v):(u, v) está em p}
7          para cada aresta (u, v) em p faça
8              se (u, v) então
9                  f(u, v) = f(u, v) + c_f(p)
10             fim
11             senão
12                 f(v, u) = f(v, u) - c_f(p)
13             fim
14         fim
15     fim
16 fim
```

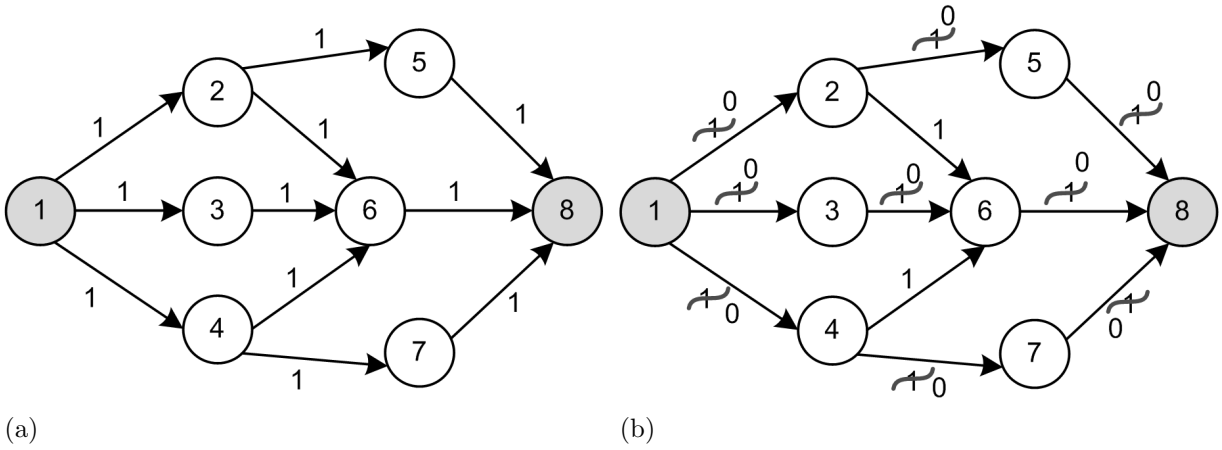
Fonte: adaptado de CORMEN et al., 2012.

3.1.3 Busca por Caminhos com Enlaces Disjuntos

O algoritmo de Edmonds e Karp permite encontrar caminhos com enlaces disjuntos em um grafo. Mas para que isso seja possível, é necessário atribuir o valor unitário à capacidade de cada aresta do grafo (CORMEN et al., 2012), como apresentado na Figura 17(a). Pois, de acordo com as propriedades de "conservação de fluxo do nó intermediário" e "restrição de capacidade" apresentadas no início deste capítulo, observada

a capacidade dos enlaces que constituem o caminho, apenas um fluxo poderá atravessar cada nó intermediário existente no mesmo.

Figura 17 - Emprego do método de Ford-Fulkerson para encontrar caminhos com enlaces disjuntos entre o nó “1” e o nó “8”.



Legenda: (a) Antes de aplicar o método de Ford-Fulkerson. A capacidade das arestas é igual a um.
(b) Após aplicar o método de Ford-Fulkerson. Algumas arestas têm capacidade igual a zero.

Ao submeter o grafo apresentado na Figura 17(a) ao algoritmo BFS, este consegue montar uma árvore formada pelos seguintes caminhos: 1-2-5-8, 1-3-6-8, 1-4-6-8, 1-2-6-8 e 1-4-7-8, ou seja, a árvore contém todos os caminhos possíveis entre o nó “1” e o nó “8”.

Como próximo passo, o algoritmo de Edmonds e Karp recebe como entrada a árvore montada pelo algoritmo BFS. Assim, o primeiro caminho é selecionado e a ele adicionado um fluxo. Consequentemente, a capacidade dos enlaces será subtraída de um e se tornará zero, conforme apresentado na Figura 17(b). Este processo será repetido com todos os caminhos encontrados pelo BFS que apresentem enlaces com capacidade igual a um, ou seja $c(u, v) = 1$. O caminho que apresentar pelo menos um enlace com capacidade igual a zero será descartado. Essa condição valerá para todos os caminhos que compartilhem enlaces entre si.

Ao fim de todo o processo, os caminhos que apresentarem todos os enlaces com capacidade zero serão considerados caminhos habilitados para o uso. Esses caminhos não compartilham enlaces entre si; ou seja, configuram caminhos constituídos por enlaces disjuntos.

O próximo capítulo trata da implementação de toda teoria sobre o teorema de fluxo máximo e corte mínimo apresentados até aqui em conjunto de instruções prontas para serem executadas por um computador. Além disso, apresenta também a implementação dos mecanismos de balanceamento de carga utilizados neste trabalho.

4 MECANISMOS DE BALANCEAMENTO DE CARGA AVALIADOS

Os documentos sobre controlador ODL disponíveis em seu domínio oficial, conferido até a data de conclusão deste trabalho, não fazem referência a existência de uma aplicação nativa e distribuída em conjunto com o controlador ODL que realize balanceamento de carga. Inclusive, em (DUQUE; BELTRáN; LEGUIZAMóN, 2018) os autores comentam que o ODL não possui um módulo padrão que permita selecionar o caminho mais adequado entre origem e destino. E que em vez disso, o ODL inunda a rede com a replicação dos fluxos.

Dessa forma, este capítulo apresenta dois mecanismos de balanceamento de carga entre enlaces avaliados neste trabalho: o MLB, proposto nesta dissertação como solução para balanceamento de carga entre enlaces utilizando controle de comutação e caminhos com enlaces disjuntos; e o de Seth, mecanismo de balanceamento de carga entre enlaces proposto por Nayan Seth que utiliza caminhos não disjuntos (SETH, 2016).

A seguir é apresentada a biblioteca de Python "NetworkX" utilizada nas implementações dos dois mecanismos para computar caminhos. Em seguida, são apresentadas as implementações de ambos os mecanismos.

4.1 Biblioteca *NetworkX*

A *NetworkX* consiste em uma biblioteca para a linguagem de programação Python que permite a exploração e a análise de estruturas semelhantes a grafos matemáticos e topologias de redes de computadores. Além disso, esta biblioteca permite representar muitos tipos de redes ou grafos, incluindo grafos simples, direcionados e grafos com arestas paralelas. Tal flexibilidade permite que essa biblioteca possa ser utilizada em vários campos científicos. Também possui uma grande diversidade de algoritmos oferecidos como funções inclusas em sua estrutura que permitem efetuar cálculos como por exemplo o de caminho mais curto entre dois pontos (HAGBERG; SCHULT; SWART, 2008).

A *NetworkX* foi utilizada para computar caminhos entre dois pontos da rede, origem e destino, nas implementações dos dois mecanismos de balanceamento de carga avaliados neste trabalho, de Seth e o MLB.

Após o processo de descoberta da topologia pelo controlador SDN, as informações de nós e enlaces são processadas e submetidas à *NetworkX*. Na computação de caminhos pelo mecanismo de Seth foi utilizada a função "all_shortest_paths()" que retorna todos os caminhos como o menor número de nós entre origem e destino e com a mesma quantidade de nós (HAGBERG; SCHULT; SWART, 2004a). Esta função utiliza o algoritmo de Dijkstra para encontrar os caminhos não disjuntos. Já no MLB, a computação de ca-

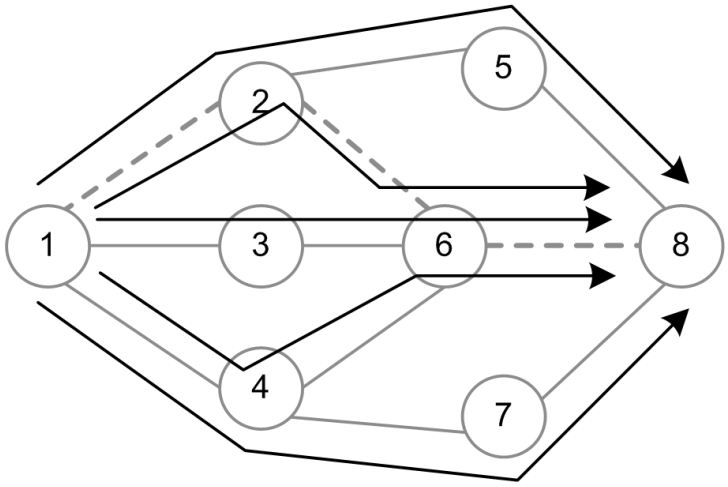
minhos utiliza a função “edge_disjoint_paths()” que retorna todos os caminhos de enlaces disjuntos com o menor número e mesma quantidade de nós entre origem e destino (HAGBERG; SCHULT; SWART, 2004b). Esta função utiliza a teoria de fluxo máximo em redes (algoritmo de Edmonds e Karp).

4.2 **Implementação de Seth**

Esta implementação baseia-se no algoritmo de Dijkstra para computar múltiplos caminhos não disjuntos que apresentem o menor número de saltos entre a origem e o destino. A escolha pelo mecanismo de Seth deu-se pelo fato de seu código fonte estar disponível em diretório público sob licença de distribuição *GENERAL PUBLIC LICENSE* (GNU) para softwares livres (SETH, 2016). Além disso, o mecanismo de Seth foi objeto de estudo da dissertação de (HASSAN, 2017), na qual realiza uma avaliação de desempenho e apresenta bons resultados para a vazão agregada na rede.

A Figura 18 apresenta um exemplo da computação de caminhos não disjuntos realizada pelo mecanismo de Seth para a topologia utilizada na avaliação de desempenho deste trabalho.

Figura 18 - Computação de caminhos entre os nós 1 e 8
realizada pelo mecanismo de Seth.



Legenda: → cinco caminhos não disjuntos.
- - - caminho padrão formado pelos nós 1-2-6-8.

O pseudocódigo desta implementação é apresentado na Figura 19. Os hosts de origem e destino são definidos no início da execução, como mostra as linhas 2 e 3. A implementação de Seth está estruturada de maneira a realizar balanceamento de carga de apenas um fluxo. Por isso, são passados apenas endereços de origem e destino. Na

linha 4 é definido o tempo de espera entre cada rodada de execução que será aplicado na linha 17. O tempo de 60 s foi definido pelo autor do algoritmo e mantido nos experimentos realizados, para que desta maneira não descaracteriza-se a obra do autor. Na linha 5 tem início a estrutura de repetição condicionada a sentença "enquanto $(1 > 0)$ faça", que mantém a aplicação em constante execução sem definição de término. Na linha 6 a função "proc_topologia()" retorna a informação das conexões estabelecidas entre os nós. Já a linha 7 carrega a biblioteca "*NetworkX*" recebendo como parâmetro de entrada as informações da topologia armazenadas em "nos_conectados". Na linha 8, a estrutura de grafos "G" é processada pela função "all_shortest", baseada no algoritmo de Dijkstra. Como resultado, são retornados todos os caminhos computados entre origem e destino e armazenados na variável "caminhos". Na linha 9, tem início um "*loop*" que se repetirá dada a quantidade de caminhos computados. Dentro do laço, na linha 10, "t1" recebe do controlador, no tempo atual, as informações do volume de dados transmitidos e recebidos pelas interfaces dos comutadores que compõem o caminho; e passados 2 s de espera, "t2" recebe também do controlador, no tempo atual, as informações do volume de dados transmitidos e recebidos pelas interfaces dos comutadores que compõem o caminho, como mostra a linha 12. Então, na linha 13 o valor de "t2" é subtraído por "t1" e o resultado armazenado no vetor "custo_caminhos". A variável "melhor_caminho" recebe o caminho com menor volume de dados registrado, na linha 15. E finalmente, na linha 16, a aplicação submete ao controlador as informações do novo caminho que o fluxo deve utilizar.

Figura 19 - Pseudocódigo da implementação do mecanismo de Seth.

```
Implementação de Seth
1  início
2      origem = IP host de origem
3      destino = IP host de destino
4      t = 60 segundos
5      enquanto (1 > 0) faça
6          nos_conectados = proc_topologia()
7          G = networkx.graph(nos_conectados)
8          caminhos = networkx.all_shortest(G, origem, destino, dijkstra)
9          para i de 1 até quantidade(caminhos) faça
10             t1 = retorna_dados_tx_rx(OpenDaylight_restconf, caminhos[i])
11             espera 2 s
12             t2 = retorna_dados_tx_rx(OpenDaylight_restconf, caminhos[i] )
13             custo_caminhos[i] = t2 - t1
14         fim
15         melhor_caminho = retorna_indice(min(custo_caminhos))
16         configura_fluxo(caminhos[melhor_caminho], origem, destino)
17         espera(t)
18     fim
19 fim
```

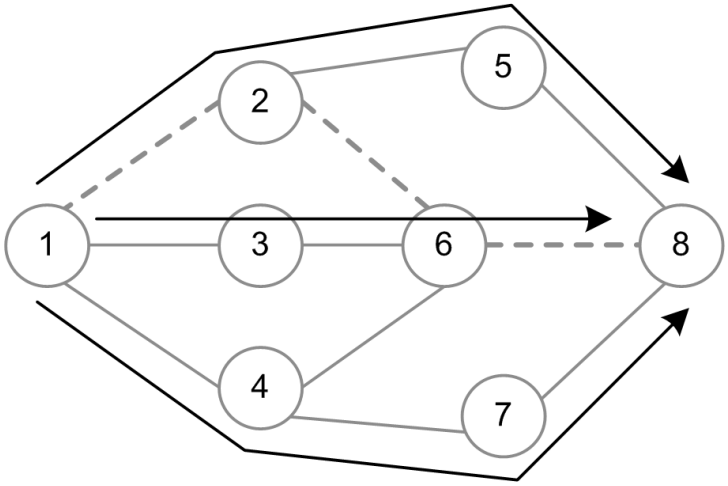
4.3 Implementação MLB

Esta implementação baseia-se no algoritmo de Edmonds e Karp (CORMEN et al., 2012) que computa múltiplos caminhos com enlaces disjuntos que apresentam o menor número de saltos entre a origem e o destino. Ela difere da implementação de Seth não somente pelo tipo de computação de caminhos, mas também pela quantidade de fluxos que pode manipular. A implementação de Seth foi codificada, por seu autor, para realizar o balanceamento de carga de apenas um único fluxo utilizando os campos de cabeçalho: IP de origem e IP de destino. O mecanismo MLB foi codificado para permitir o balanceamento de carga por mais de um fluxo utilizando os campos de cabeçalho: IP de origem, IP de destino, protocolo e porta de destino. A quantidade de fluxos balanceados pelo MLB é parametrizado no código fonte de sua implementação. Para os experimentos realizados neste trabalho foram parametrizados: um fluxo, para tornar justa a comparação de desempenho com o mecanismo de Seth; e três fluxos, para avaliar o desempenho do balanceamento de carga com mais de um fluxo. Além disso, diferentemente da implementação de Seth que comuta caminhos a cada 60 s, o MLB introduz um "controle de comutação" que verifica se a ocupação atual do caminho ultrapassa 50% da capacidade deste e se o potencial novo caminho computado apresenta uma ocupação pelo menos 10% menor do

que a do caminho atual.

A Figura 20 apresenta um exemplo de computação de caminhos com enlaces disjuntos realizada pelo mecanismo de MLB para a topologia utilizada na avaliação de desempenho deste trabalho.

Figura 20 - Computação de caminhos entre os nós 1 e 8
realizada pelo mecanismo de MLB.



Legenda: → três caminhos com enlaces disjuntos.
- - - caminho padrão formado pelos nós 1-2-6-8.

O pseudocódigo desta implementação é apresentado na Figura 21. As linhas 2 e 3 definem os comutadores de origem e destino. Já a linha 4, define a quantidade de fluxos que serão balanceados. Nas linhas 5 , 6 e 7 são definidos os valores de referência para uso pelo controle de comutação, que pode ser visto na linha 21 e será explicado mais abaixo. Na linha 8 tem início a estrutura de repetição condicionada a sentença "enquanto $(1 > 0)$ faça", que mantém a aplicação em constante execução sem definição de término. Na linha 9 a função "proc_topologia()" retorna a informação das conexões estabelecidas entre os nós. Já a linha 10 carrega a biblioteca "*NetworkX*" recebendo como parâmetro de entrada as informações da topologia armazenadas em "nos_conectados". Já a linha 11 apresenta estrutura de repetição condicionada ao número de fluxos em uso; ou seja, todo processo é repetido para cada fluxo em uso. Na linha 12, a estrutura de grafos "G" é processada pela função "edge_disjoint()" utilizando o método de computação de caminhos de Edmonds e Karp. Como resultado, são retornados os caminhos computados e armazenados na variável "caminhos". Na linha 13, tem início um "*loop*" que se repetirá dada a quantidade de caminhos computados. Neste, a função "calcula_vazao" estima as vazões nos caminhos computadas pelas informações de volume de dados transmitidos e recebidos pelos comutadores que compõem cada caminho, sendo armazenadas em "custo_caminhos", na linha 14. Na linha 16, a variável "melhor_caminho" recebe o caminho com menor vazão

computada. E por fim, A linha 17, testa se é a primeira rodada de execução da aplicação para o fluxo corrente. Em caso positivo, o caminho para o fluxo é configurado na linha 18 e o vetor "atual" recebe o índice do melhor caminho para o fluxo corrente, conforme apresentado na linha 19. Em caso negativo, a linha 21 apresenta o controle de comutação que permite a comutação do fluxo para o novo caminho caso a ocupação do caminho atual seja maior ou igual ao valor definido na variável "condicao1", neste caso 50% da sua capacidade, e o novo caminho apresente uma ocupação menor em 10% ou mais da ocupação do caminho atual, valor definido na variável "condicao2". Para os casos em que as duas condições forem satisfeitas o novo caminho alternativo será configurado para o fluxo corrente na linha 22 e o vetor "atual" receberá o índice do melhor caminho para o fluxo corrente, conforme apresentado na linha 23.

Figura 21 - Pseudocódigo da implementação do mecanismo MLB.

```
    Implementação MLB
 1  início
 2      origem = ID switch de origem
 3      destino = ID switch de destino
 4      qtd_fluxo = quantidade de fluxo
 5      condicao1 = 50%
 6      condicao2 = 10%
 7      capacidade_caminho = 10.000.000
 8      enquanto (1 > 0) faça
 9          nos_conectados = proc_topologia()
10          G = networkx.graph(nos_conectados)
11          para fluxo de 1 até qtd_fluxos faça
12              caminhos = networkx.edge_disjoint(G, origem, destino, edmonds_karp)
13              para i de 1 até quantidade(caminhos) faça
14                  custo_caminhos[i] = calcula_vazao(OpenDaylight_restconf, caminhos[i])
15              fim
16              melhor_caminho = retorna_indice(min(custo_caminhos))
17              se atual[fluxo] = ∅ então
18                  configura_fluxo(caminhos[melhor_caminho], origem, destino, fluxo)
19                  atual[fluxo] = melhor_caminho
20              fim
21              senão se custo_caminhos[atual[fluxo]] maior em (condicao1) ou mais que
                  capacidade_caminho & custo_caminhos[melhor_caminho]
                  menor em (condicao2) ou mais que custo_caminhos[atual[fluxo]] então
22                  configura_fluxo(caminhos[melhor_caminho], origem, destino, fluxo)
23                  atual[fluxo] = melhor_caminho
24              fim
25          fim
26      fim
27  fim
```

5 AVALIAÇÃO DE DESEMPENHO

Este capítulo apresenta uma avaliação de desempenho utilizando o controlador ODL desprovido do uso de mecanismos de balanceamento de carga, com o uso do mecanismo de Seth e com o uso do mecanismo MLB. São apresentadas as ferramentas, modelos de tráfego, mecanismos e métricas empregadas na avaliação de desempenho. Além disso, são apresentados também os resultados obtidos durante a condução dos experimentos.

5.1 Ferramentas Utilizadas

Esta seção apresenta as ferramentas que foram utilizadas para criar o ambiente de rede e gerar tráfego de dados no mesmo.

5.1.1 Emulador Mininet

A condução de experimentos em redes de computadores resulta na necessidade de se montar uma infraestrutura que possa permitir no mínimo a comunicação entre dois *hosts*. Porém, quando o experimento tem a necessidade de uso de um número maior de *hosts*, a montagem de uma infraestrutura com dispositivos reais necessários à reprodução do ambiente computacional de maneira confiável implica dificuldades como dispor de espaço físico para reunir todos os equipamentos da rede, dispor de pontos de alimentação elétrica para todos os dispositivos da rede, dispor de computadores que assumirão os papéis dos *hosts*, montar a estrutura de cabeamentos com comutadores responsáveis pela interconexão dos hosts da rede, entre outras dificuldades. Além disso, o experimento pode ser tornar financeiramente inviável. Dessa forma, o uso de uma ferramenta que emule o ambiente de rede computacional mostra-se uma boa opção para contornar tais dificuldades.

O Mininet consiste em um sistema de emulação que permite prototipar topologias de redes SDN, mesmo dispondo de poucos recursos computacionais (LANTZ; HELLER; MCKEOWN, 2010). Uma característica importante do Mininet é o fato de permitir o uso de comutadores OpenFlow baseados em virtualização no nível de processos do *kernel* do sistema operacional. Por meio de uma API, o Mininet pode receber instruções de comando via *scripts* codificados na linguagem de programação Python e facilmente emular uma rede SDN com dezenas de comutadores em um único computador (KREUTZ et al., 2015).

A avaliação de desempenho foi realizada em um computador equipado com processador Intel Core i7-4770 de oito núcleos com 3,4 GHz e 8 GB de memória RAM, utilizando

a versão 16.04 do sistema operacional Ubuntu.

5.1.2 iPerf

Todos os fluxos de dados utilizados na avaliação de desempenho foram gerados pela ferramenta iPerf. Trata-se de um programa de computador que permite gerar e medir tráfego de dados em redes de computadores. O iPerf é baseado na arquitetura de *software* cliente e servidor, na qual o cliente gera as solicitações e o servidor as responde. Não possui interface gráfica e por padrão funciona a partir da linha de comando dos computadores pertencentes à rede.

O iPerf executa um cliente em um computador definido como ponto de origem na rede e um servidor em um computador definido como ponto de destino na rede. Em seguida, o cliente gera pacotes de dados e os transmite ao servidor utilizando os protocolos de transporte TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*). O servidor, ao receber os pacotes, analisa os dados e gera relatórios de resultados como: largura de banda, perda de pacotes, tempo de resposta e *jitter* (SILVA; JUNIOR, 2014).

5.1.3 CURL

O *Client URL* (CURL) trata-se de uma ferramenta de linha de comando utilizada para transferir informações encapsuladas de modelos de transferência de dados, como por exemplo JavaScript Object Notation (JSON) e Extensible Markup Language (XML).

Por meio do CURL pode-se realizar a transferência das políticas de tráfego definidas para a rede ao controlador SDN. Tais políticas são modeladas pelos administradores de rede no formato de dados XML e carregadas no controlador por meio da interação do CURL com a REST API do ODL (Northbound API utilizada pelo ODL). O controlador ao receber as políticas utiliza o protocolo OpenFlow para estabelecer a comunicação com os comutadores por meio da Southbound API. Por fim, o controlador aplica as políticas de tráfego traduzidas em regras de fluxos aos comutadores da rede.

5.2 **Métricas**

O desempenho dos mecanismos de balanceamento de carga foram avaliados com base em três métricas:

a) vazão agregada: a utilização de balanceamento de carga tende a aumentar a largura de banda agregada. Essa métrica é constituída pelo so-

matório das vazões dos fluxos observados no destino. Para o seu cálculo, foram utilizados fluxos compostos por pacotes TCP;

b) perda de pacotes: a perda de pacotes é decorrente principalmente do congestionamento no caminho. A medição é feita no destino com base na diferença entre a quantidade de pacotes que foi enviada pela origem e o que foi recebido no destino. Neste caso foram utilizados fluxos compostos por pacotes UDP;

c) justiça: o uso de um caminho por mais de um fluxo é considerado justo quando todos os fluxos fazem uso de partes iguais da largura de banda disponível. O índice de justiça é uma métrica que retorna um valor compreendido entre 0 e 1. Se todos os fluxos tiverem uma vazão praticamente igual entre eles, o resultado se manterá próximo a 1. Se ocorrerem diferenças significativas nas taxas de vazão, o resultado tenderá a 0. O índice de justiça (JAIN, 1991) é dado por:

$$f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2},$$

onde "n" é o número total de fluxos concorrendo pela largura de banda e "x_i" é o valor da vazão do fluxo "i". Para o seu cálculo, foram utilizados fluxos compostos por pacotes TCP.

5.3 **Modelos de Tráfego**

Em redes SDN, as decisões de encaminhamento são baseadas em fluxos de dados, que são definidos de acordo com alguns campos de seu cabeçalho apresentados na Figura 4. Dessa forma, optou-se por utilizar os campos de "IP origem", "IP destino", "tipo de protocolo" e "porta de destino" para identificar os fluxos de dados gerados. Além disso, de maneira a conseguir gerar outras cargas na rede e ainda poder avaliar os benefícios do balanceamento de carga, foi definido um fluxo de fundo para ser transmitido juntamente com os fluxos principais. Assim, foram configurados quatro modelos de tráfego:

a) Modelo TCP-2f: composto por dois fluxos, sendo o fluxo principal tendo como origem o host "h1" e como destino o host "s5" na porta TCP/5001 e o fluxo de fundo tendo como origem o host "h2" e "s6" como destino na porta TCP/5002 (ver Figura 22). A duração dos fluxos foi fixada em

600 s. Esse modelo foi utilizado para medir a vazão agregada e o índice de justiça no caso de dois fluxos;

b) Modelo UDP-2f: composto por dois fluxos, sendo o fluxo principal tendo como origem o host “h1” e como destino o host “s5” na porta UDP/5001 e o fluxo de fundo tendo como origem “h2” e como destino “s6” na porta UDP/5002 (ver Figura 22). A duração dos fluxos foi fixada em 600 s. Esse modelo foi utilizado para medir a perda de pacotes para o caso de dois fluxos;

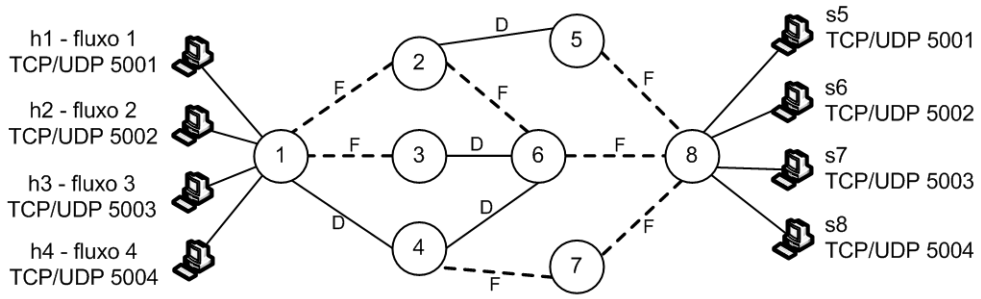
c) Modelo TCP-4f: composto por quatro fluxos, sendo três deles fluxos principais com origem, destino e porta definidos da seguinte forma: fluxo 1, origem host “h1” e destino host “s5” na porta TCP/5001; fluxo 2, origem host “h2” e destino host “s6” na porta TCP/5002; e fluxo 3, origem host “h3” e destino host “s7” na porta TCP/5003. O fluxo de fundo teve como origem o host “h4” e destino o host “s8” na porta TCP/5004 (ver Figura 22). A duração dos fluxos foi fixada em 600 s. Esse modelo foi utilizado para medir a vazão agregada e o índice de justiça quando o número de fluxos é igual a quatro;

d) Modelo UDP-4f: composto por quatro fluxos, sendo três deles fluxos principais com origem, destino e porta definidos da seguinte forma: fluxo 1, origem host “h1” e destino host “s5” na porta UDP/5001; fluxo 2, origem host “h2” e destino host “s6” na porta UDP/5002; e fluxo 3, origem host “h3” e destino host “s7” na porta UDP/5003. O fluxo de fundo foi definido tendo como origem o host “h4” e destino o host “s8” na porta UDP/5004 (ver Figura 22). A duração dos fluxos foi fixada em 600 s. Esse modelo foi utilizado para medir a perda de pacotes dos quatro fluxos.

A Figura 22 apresenta a topologia utilizada nos experimentos. Ela possui oito nós, três caminhos com enlaces disjuntos e cinco caminhos não disjuntos. A capacidade de todos os enlaces da rede foi configurada em 10 Mbps devido aos recursos computacionais disponíveis.

Os fluxos UDP foram configurados para transmitirem dados a uma taxa de 9,5 Mbps, ou seja, ocupando 95% da capacidade dos enlaces da rede. Essa taxa de transmissão foi encontrada após a execução de vários testes nos quais a perda de pacotes foi medida. A taxa de transmissão foi aumentada sucessivamente até que se chegasse ao limite de uma transmissão sem perda de pacotes.

Figura 22 - Topologia utilizada na avaliação de desempenho com os fluxos utilizados pelos modelos de tráfego.



Legenda: (D) "Discarding", enlace desabilitado.
(F) "Forwarding", enlace habilitado.
(- - -) caminho padrão definido pela árvore de cobertura montada pelo módulo *loop remover* da aplicação l2switch do ODL.

5.4 **Mecanismos Avaliados**

A execução dos experimentos foi dividida em duas partes: experimento "com dois fluxos", composto pelos modelos de tráfego com dois fluxos (TCP-2f e UDP-2f) e experimento "com quatro fluxos", composto pelos modelos de tráfego com quatro fluxos (TCP-4f e UDP-4f). Cada experimento teve os seus modelos de tráfego submetidos ao funcionamento dos três mecanismos detalhados a seguir:

a) ODL: é o controlador ODL padrão desprovido de balanceamento de carga; ou seja, todos os fluxos são comutados pelo mesmo caminho padrão definido na árvore de cobertura. Desta forma, as regras de fluxo para o devido encaminhamento do fluxo principal quanto do fluxo de fundo serão configuradas manualmente obedecendo o caminho padrão definido pela árvore de cobertura;

b) de Seth: realiza balanceamento de carga utilizando a implementação de Seth baseada em caminhos não disjuntos. De acordo com a implementação codificada por seu autor, este mecanismo realiza o balanceamento de carga de apenas um fluxo, neste caso do fluxo principal originado pelo host "h1". Desta forma, durante o uso dos modelos de tráfego com dois fluxos serão configuradas manualmente as regras de fluxo para o devido encaminhamento do fluxo de fundo, originado no host "h2", obedecendo o caminho padrão. Porém, durante o uso dos modelos de tráfego com quatro fluxos serão configuradas manualmente as regras de fluxo para o devido encaminhamento do fluxo de fundo, originado no

host “h4” e dos fluxos principais originados nos hosts “h2” e “h3”.

c) MLB: realiza balanceamento de carga utilizando a implementação MLB baseada em caminhos com enlaces disjuntos. Este mecanismo realiza balanceamento de carga de um ou mais fluxos, conforme definição de parâmetro estabelecida em seu código fonte. Desta forma, as regras de fluxo para o devido encaminhamento do fluxo de fundo serão configuradas manualmente obedecendo o caminho padrão definido pela árvore de cobertura;

A justificativa para se dividir o experimento em duas partes ocorre pelo fato do mecanismo de Seth realizar balanceamento de carga de apenas um dos fluxos; neste caso do fluxo principal. Já o mecanismo MLB pode realizar o balanceamento de mais fluxos. Contudo, para tornar justa esta avaliação, os dois mecanismos foram submetidos aos dois experimentos e aos seus modelos de tráfego.

Para cada modelo de tráfego foram executadas 30 rodadas. Os resultados foram processados com base no uso de média aritmética e intervalo de confiança de 95%.

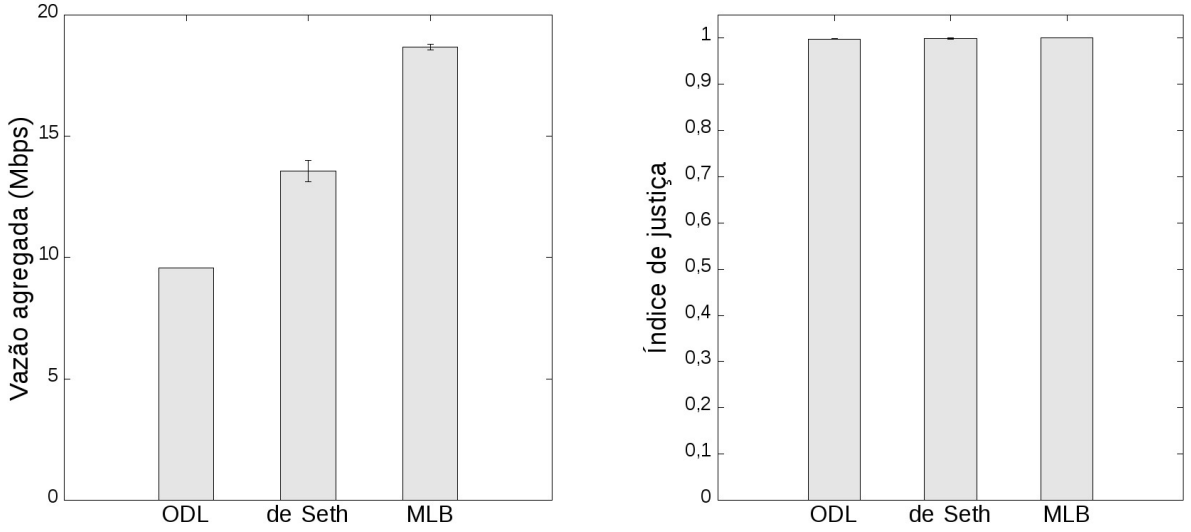
5.5 Resultados

Experimento com dois fluxos

A Figura 23(a) apresenta os resultados da vazão agregada utilizando o modelo de tráfego TCP-2f. Nesta é possível observar que utilizando apenas o ODL, a vazão agregada é de aproximadamente 10 Mbps. Neste caso, o fluxo principal e o fluxo de fundo trafegam juntos dividindo a largura de banda do caminho padrão montado pelo ODL (árvore de cobertura) e formado pelos nós 1-2-6-8, conforme mostrado na Figura 24. Assim, como não existe balanceamento de carga, o valor da vazão agregada será menor ou igual a largura de banda do caminho padrão.

No balanceamento de Seth, a vazão agregada apresenta um considerável aumento de 41% em relação ao ODL. Isso porque o balanceamento de Seth realiza a comutação do fluxo principal para um novo caminho alternativo que apresente menor volume de tráfego computado periodicamente. O fluxo de fundo não sofre balanceamento de carga e segue pelo caminho padrão formado pelos nós 1-2-6-8, já o fluxo principal é comutado ora para um caminho sem enlaces compartilhados pelo fluxo de fundo, formado pelo nós 1-4-7-8, ora para o caminho padrão utilizado pelo fluxo de fundo e ora por caminhos com enlace compartilhado com o caminho padrão, formados pelos nós 1-2-5-8, 1-3-6-8 e 1-4-6-8, como apresentado na Figura 24. Neste caso, com o balanceamento de carga, a vazão agregada alcançou 13,5 Mbps.

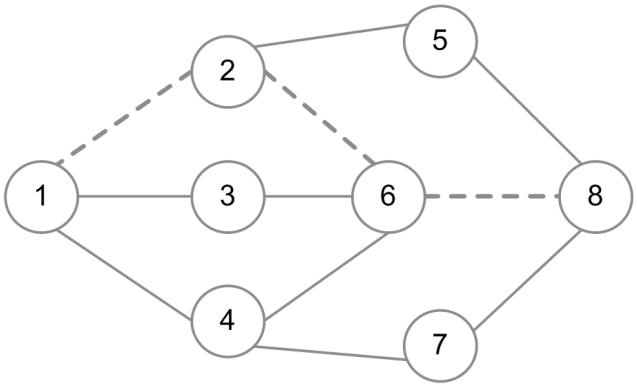
Figura 23 - Experimento com dois fluxos - resultados da vazão agregada e índice de justiça.



(a) Vazão agregada com uso do protocolo TCP.

(b) Índice de justiça com uso do protocolo TCP.

Figura 24 - Caminho padrão entre os nós 1 e 8 definido pela árvore de cobertura montada pelo ODL.



Legenda: --- caminho padrão formado pelos nós 1-2-6-8.

Por último, no balanceamento MLB é possível atestar que a vazão agregada chegou a 18,6 Mbps, o que significa um ganho de 37% em relação a vazão agregada alcançada por de Seth e 95% em relação ao ODL. Esse melhor desempenho é atribuído ao controle de comutação que somente permite a comutação do fluxo principal quando 50% ou mais da largura de banda do seu caminho está ocupada e o novo caminho apresenta uma largura de banda ocupada inferior a 10% ou mais em relação à ocupação da largura de banda do caminho atual. Isso faz com que comutações para novos caminhos com a banda já completamente ocupada não ocorram. Também evita-se que ocorra comutação para um novo caminho que apresente nível de ocupação semelhante ao do atual caminho em uso,

o que resultaria na divisão da largura de banda do novo caminho ao invés do aumento da largura de banda agregada pelo uso de caminhos alternativos com enlaces ociosos.

A Figura 23(b) apresenta os resultados do índice de justiça utilizando o modelo de tráfego TCP-2f. Com o uso apenas do ODL, a disputa pelo uso da largura de banda entre os fluxos resultou em um índice de justiça de 0,99, o que caracteriza que a disputa pela ocupação da banda do caminho padrão ocorreu de maneira semelhante por ambos os fluxos. A taxa média de vazão do fluxo de fundo alcançou 4,8 Mbps contra 4,7 Mbps do fluxo principal.

No balanceamento de carga de Seth, o índice de justiça apresentou o mesmo valor de 0,99. O que demonstra que os fluxos ocuparam a banda de forma semelhante, mesmo ocorrendo a comutação de caminho gerada pelo balanceamento de carga. Isso significa que nos momentos em que o fluxo de fundo e o fluxo principal ocuparam juntos a banda do caminho padrão, a vazão dos fluxos se manteve aproximadamente igual e nos momentos em que a comutação do fluxo principal se deu por um caminho alternativo ambos os fluxos ocuparam praticamente a mesma largura de banda. A taxa média de vazão do fluxo de fundo alcançou 6,8 Mbps contra 6,7 Mbps do fluxo principal balanceado.

No balanceamento MLB, o resultado do índice de justiça também apresentou o valor de 0,99. A disputa entre os fluxos pela ocupação da largura de banda ocorreu de maneira semelhante. A taxa média de vazão do fluxo de fundo alcançou 9,3 Mbps contra 9,3 Mbps do fluxo principal balanceado.

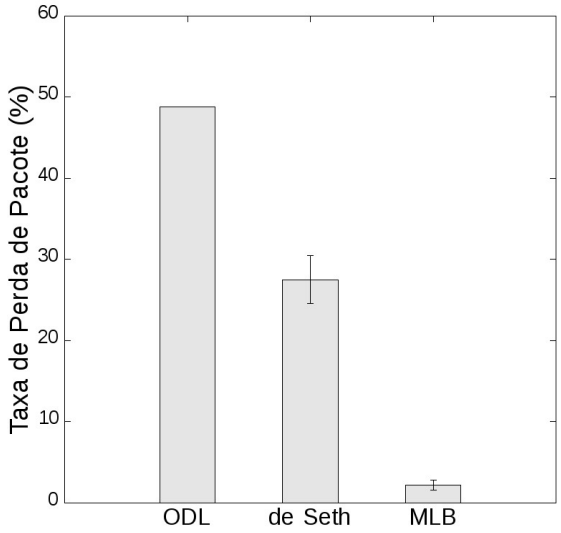
Por fim, a Figura 25 apresenta os resultados da perda de pacotes utilizando o modelo de tráfego UDP-2f. Nesta é possível observar que com o uso do ODL a perda de pacotes chega perto de 50%. Esse alto valor pode ser explicado pelo fato dos fluxos trafegarem juntos pelo caminho padrão formado pelos nós 1-2-6-8 (ver Figura 24. A largura de banda do caminhos padrão é disputada pelos dois fluxos que por serem UDP não têm controle de fluxo nem controle de congestionamento. Assim, os dois hosts de origem transmitem os fluxos a uma taxa constante de 9,5 Mbps, o que leva à grande perda de pacotes.

No balanceamento de Seth, a perda de pacotes chega perto de 27%, pois a perda ocorre apenas quando o fluxo principal balanceado divide o caminho padrão com o fluxo de fundo ou utiliza algum caminho alternativo com enlace compartilhado pelo caminho padrão, como por exemplo pelos caminhos formados pelos nós 1-2-5-8, 1-3-6-8 e 1-4-6-8 (ver Figura 24. Com a comutação do fluxo principal para o caminho formado pelos nós 1-4-7-8, os fluxos passam a não disputar mais banda entre si e a perda de pacotes cessa.

No balanceamento MLB a perda de pacotes fica abaixo de 5%. Isso ocorre, porque o controle de comutação somente permite que ocorra a comutação do fluxo principal quando houver uma diferença de 10% ou mais na taxa de ocupação do caminho atual quando comparado ao novo caminho. Quando os caminhos estão com suas larguras de banda ocupadas com níveis de carga parecidos, a comutação não ocorre. Desta forma,

no caso em que o fluxo principal segue por um caminho que não compartilhe nenhum enlace com o caminho padrão, como por exemplo o caminho formado pelos nós 1-4-7-8, a ocupação dos caminhos ocorrerá de maneira semelhante. Pois, com a atuação do controle de comutação os fluxos permanecerão por mais tempo seguindo por caminhos distintos sem que ocorra nenhuma comutação. E desta forma, por não existir disputa de banda entre os fluxos, mais pacotes são entregues com sucesso ao seu destino e consequentemente a perda de pacotes diminui. O MLB apresentou uma perda de pacotes 13 vezes menor em relação ao de Seth e 24 vezes menor em relação ao ODL.

Figura 25 - Experimento com dois fluxos - resultado da perda de pacotes com uso do protocolo UDP.



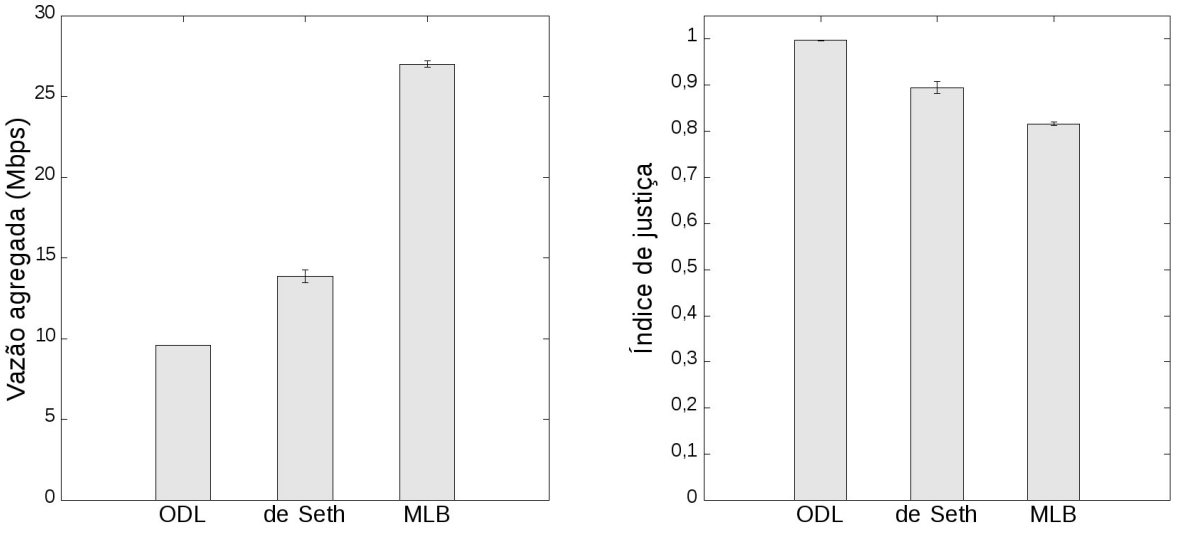
Experimento com quatro fluxos

A Figura 26(a) apresenta os resultados da vazão agregada para o modelo de tráfego TCP-4f. Nesta é possível observar que a vazão agregada utilizando apenas o ODL ficou abaixo de 10 Mbps. Sem balanceamento de carga, os três fluxos principais e o fluxo de fundo são encaminhados pelo caminho padrão formado pelos nós 1-2-6-8. Desta forma, os quatro fluxos disputam a largura de banda do caminho padrão.

No mecanismo de Seth, o balanceamento de carga permite que a vazão agregada chegue perto dos 14 Mbps. Neste mecanismo, apenas um dos fluxos principais é balanceado, os demais fluxos principais não balanceados seguem encaminhados pelo caminho padrão formado pelos nós 1-2-6-8 juntamente com o fluxo de fundo, conforme pode ser visto na Figura 27. Além disso, durante o balanceamento também ocorre do fluxo princi-

pal balanceado ser comutado para o caminho padrão dando início à disputa pela largura de banda com os demais fluxos. Isso ocorre pois a condição para que ocorra o balanceamento de carga do fluxo principal consiste em verificar qual dos caminhos apresentou menor taxa de volume de dados transmitida.Desta forma, qualquer mínima variação nos valores do volume de dados computados entre os caminhos faz com que o mecanismo de Seth execute a comutação entre eles, situação esta que tende a ocorrer com grande frequência.

Figura 26 - Experimento com quatro fluxos - resultados da vazão agregada e índice de justiça.

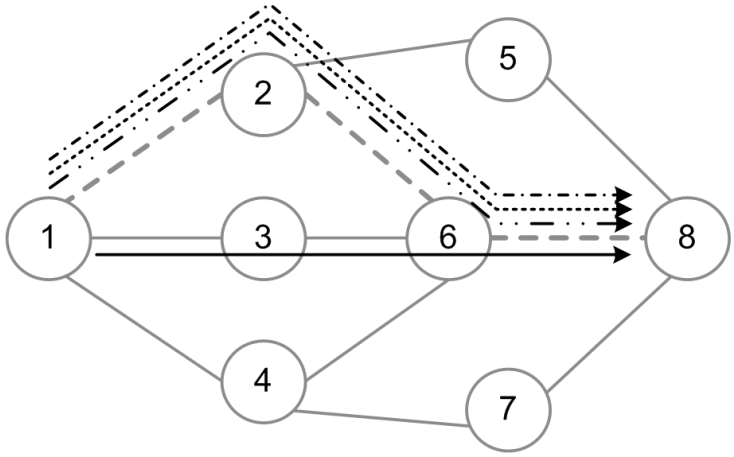


(a) Vazão agregada com uso do protocolo TCP.

(b) Índice de justiça com uso do protocolo TCP.

No mecanismo MLB, a vazão agregada alcançou os 27 Mbps. Este resultado pode ser atribuído aos seguintes fatos: no mecanismo MLB o balanceamento de carga é aplicado aos três fluxos principais; os caminhos computados não compartilham enlaces, o que evita que ocorra disputa de banda com fluxos encaminhados por outros caminhos; o controle de comutação evita que ocorram comutações quando não atendidas suas condições de funcionamento (ver Seção 4.3). Um dos fluxos principais apresentou uma taxa média de vazão de 8,8 Mbps, os outros dois fluxos principais apresentaram uma taxa média de vazão de 8,5 Mbps e o fluxo de fundo de 1,2 Mbps. Conforme o esquema de comutação de caminhos apresentado na Figura 28, assuma que o primeiro fluxo principal foi comutado pelo caminho formado pelos nós 1-4-7-8. O segundo fluxo principal foi comutado pelo caminho formado pelos nós 1-3-6-8, realizando disputa de banda com o fluxo de fundo no enlace composto pelos nós 6-8. E o terceiro fluxo principal foi comutado pelo caminho formado pelos nós 1-2-5-8, realizando disputa com o fluxo de fundo no enlace formado pelos nós 1-2. Com essa configuração de fluxos e os valores de vazão média apresentados, pode-se assumir também que o controle de comutação evitou a execução de comutações, já que a condição que estabelece que a taxa de ocupação do novo caminho deva ser

Figura 27 - Comutação do fluxo principal para o caminho 1-3-6-8 utilizando o mecanismo de Seth.



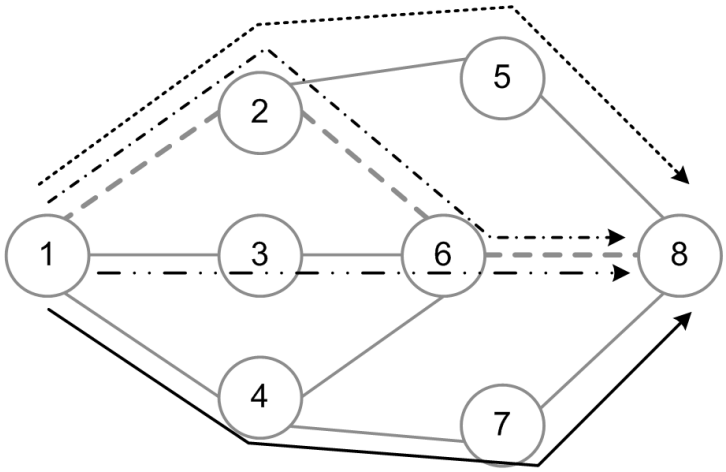
Legenda: ····· Fluxo principal não balanceado;
– · – Fluxo principal não balanceado;
—— Fluxo principal balanceado;
-·-·-·- Fluxo de fundo;
------ caminho padrão.

inferior a 10% ou mais em relação a ocupação do caminho atual não foi atendida. Assim, os fluxos permaneceram por mais tempo sendo encaminhados nesta configuração, o que proporcionou um alto valor de vazão agregada. Cabe ressaltar, que sua vazão agregada supera em 181% a vazão agregada apresentada pelo ODL e em 94% a vazão agregada apresentada pelo mecanismo de Seth.

A Figura 26(b) apresenta os resultados do índice de justiça para o modelo de tráfego TCP-4f resultante da disputa de largura de banda dos caminhos pelos fluxos. O mecanismo ODL apresentou o valor de 0,99, o que significa que os quatro fluxos ocuparam de forma semelhante a largura de banda do caminho padrão. A taxa média de vazão de cada fluxo chegou a 2,3 Mbps.

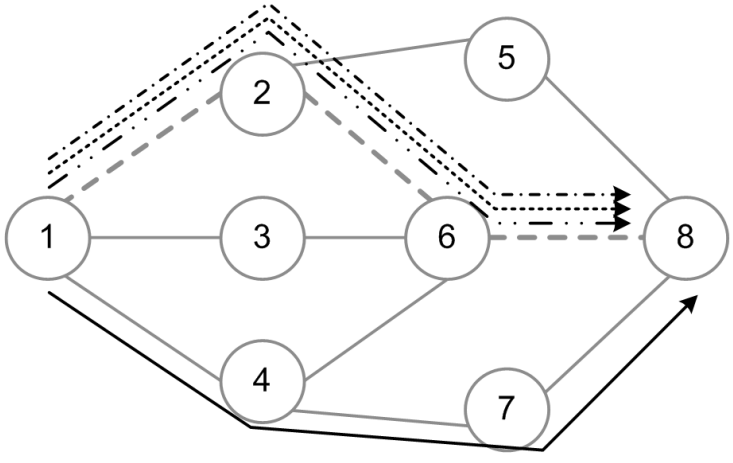
O uso de balanceamento de carga de Seth resultou em uma queda no índice de justiça que chegou próximo de 0,89. Isso pode ser justificado pelo fato de realizar o balanceamento de carga de apenas um fluxo principal, os demais fluxos são encaminhados pelo caminho padrão formado pelos nós 1-2-6-8. Quando o fluxo balanceado divide a largura de banda com os demais fluxos, seja por estar no caminho padrão ou seja por estar em um caminho que possua enlace compartilhado, a ocupação de banda entre os fluxos tende a ser igual. Porém, quando o fluxo balanceado é comutado para um caminho no qual não exista compartilhamento de enlace pelos demais fluxos, como por exemplo pelo caminho formado pelos nós 1-4-7-8, conforme mostrado na Figura 29, o fluxo balanceado tende a ocupar toda a banda deste caminho. Com isso, sua taxa de ocupação de banda

Figura 28 - Comutação dos fluxos principais utilizando o mecanismo MLB.



Legenda: ····· Fluxo principal balanceado;
– · – Fluxo principal balanceado;
—— Fluxo principal balanceado;
-·-·- Fluxo de fundo;
------ caminho padrão.

Figura 29 - Comutação do fluxo principal para o caminho 1-4-7-8 utilizando o mecanismo de Seth.



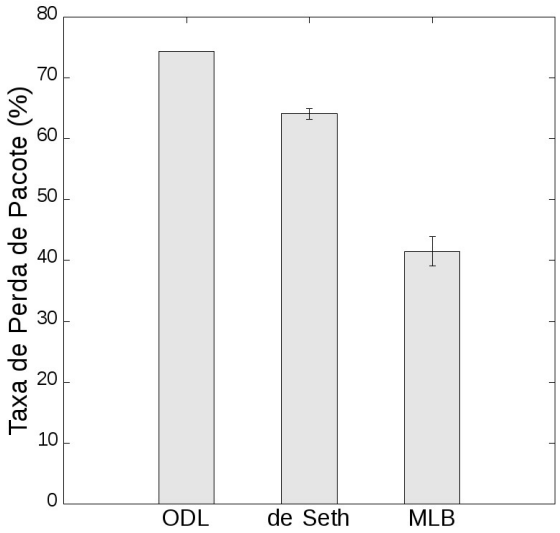
Legenda: ····· Fluxo principal não balanceado;
– · – Fluxo principal não balanceado;
—— Fluxo principal balanceado;
-·-·- Fluxo de fundo;
------ caminho padrão.

torna-se maior em relação a alcançada pelos demais fluxos que seguem dividindo a largura de banda do caminho padrão. Neste caso, os três fluxos apresentaram vazões semelhantes de 2,8 Mbps. Enquanto que o fluxo principal balanceado alcançou uma vazão média de 5,5 Mbps.

No balanceamento de carga MLB, o índice de justiça alcançou o valor de 0,81. Isso porque, apesar do balanceamento de carga dos três fluxos principais permitir que cada fluxo seja encaminhado por um dos três caminhos formados por enlaces disjuntos, o fluxo de fundo encaminhado pelo caminho padrão consome a largura de banda de dois dos caminhos com enlaces disjuntos computados pelo MLB, os caminhos 1-2-5-8 e 1-3-6-8 (ver Figura 28). Dessa forma, os fluxos que forem comutados para este dois caminhos disputarão a largura de banda com o fluxo de fundo encaminhado pelo caminho padrão. Neste caso, o fluxo de fundo apresentou uma vazão média de 1,2 Mbps, enquanto que os fluxos balanceados apresentaram uma vazão média de 8,5 Mbps.

A Figura 30 apresenta os resultados de perda de pacotes para o modelo de tráfego UDP-4f. Os resultados apresentados nesta figura são bem semelhantes aos obtidos no experimento com dois fluxos, porém com maiores valores de perdas, já que agora são transmitidos quatro fluxos. A diferença dos resultados entre o ODL e mecanismo de Seth diminuiu. O ODL chegou a 74% de perda de pacotes e o mecanismo de Seth apresentou perda próxima de 65%.

Figura 30 - Experimento com quatro fluxos - resultado da perda de pacotes com uso do protocolo UDP.



Isso provavelmente ocorreu devido ao congestionamento causado pela transmissão simultânea dos quatro fluxos. O mecanismo de Seth acaba apresentando um comportamento muito parecido com o ODL. Isso porque, como já mencionado, no mecanismo de

Seth apenas um fluxo principal é balanceado; os demais são transmitidos pelo caminho padrão, semelhantemente ao ODL. Já a taxa de perda de pacotes do MLB chegou perto de 42%. Isso porque os três fluxos principais são comutados para caminhos menos ocupados. O balanceamento de carga e o controle de comutação do MLB permitem que mais pacotes sejam entregues a seus destinos por caminhos menos congestionados.

CONCLUSÃO

A compreensão de que a união de redes *Ethernet* realizada pela interligação de múltiplos comutadores podem implicar em enlaces ociosos, dependendo do seu arranjo topológico, não é uma nova descoberta da área de redes. Na verdade, trata-se de uma característica deste tipo de configuração de rede causada pelo uso do protocolo STP, que desabilita determinados enlaces para evitar que os pacotes de dados permaneçam sendo propagados pela rede de modo a gerarem *loops* na rede. Esse funcionamento vai de encontro com a necessidade atual dos usuários, que buscam cada vez mais o consumo de dados e serviços ofertados nas redes se valendo de boas taxas de vazão de dados.

Diante do cenário exposto, este trabalho buscou propor um mecanismo de balanceamento de carga entre caminhos implementado juntamente com um controle de comutação utilizando SDN. Acompanhando a tendência global de consumo de serviços na Internet e dentro dos próprios domínios de redes corporativas, acadêmicas e até mesmo domésticos, o uso de SDN permite que administradores de rede desenvolvam aplicações customizadas de maneira a facilitar as rotinas de manutenção e operação das redes de computadores. E consequentemente, ajudam a diminuir os gastos com aquisição de caixas fechadas com funções especializadas (it appliances).

O mecanismo MLB, mostrou-se uma boa solução para evitar a ociosidade de enlaces causada pelo protocolo STP e consequentemente contribuiu para o aumento da largura de banda agregada na rede. Além disso, foi realizada uma avaliação de desempenho focada em apresentar os resultados da vazão agregada, da perda de pacotes e da justiça entre fluxos na rede. Foram comparados os mecanismos MLB e de Seth com o uso do ODL puro. O mecanismo MLB apresentou melhores resultados em relação ao modo de funcionamento padrão do ODL e ao mecanismo de Seth. Este melhor desempenho pode ser atribuído ao uso da computação de caminhos com enlaces disjuntos e ao controle de comutação realizado com base na ocupação do caminho pela carga de dados, ambos implementados no MLB. A utilização de caminhos com enlaces disjuntos pode evitar o compartilhamento da largura de banda pelos fluxos e consequentemente levar à diminuição da perda de pacotes. Já o controle de comutação, evita que ocorram comutações de fluxos para caminhos com níveis de ocupação saturados e que possam causar congestionamento na rede ao invés de evitá-lo.

Um ponto importante e que merece destaque, se dá quanto a característica de funcionamento no modo proativo pela aplicação *l2switch* do controlador de rede OpenDaylight. O seu funcionamento gera grande volume de dados na rede em decorrência da replicação dos fluxos. Como solução encontrada, foi realizada a configuração manual de regras de fluxos específicas com objetivo de encaminhar o tráfego de fundo pelo caminho padrão da rede definido pela árvore de cobertura.

Por fim, mesmo o MLB apresentado melhores resultados em seu desempenho frente aos outros dois mecanismos, cabe salientar, que de modo geral, os três mecanismos avaliados apresentaram um bom desempenho.

Direções Futuras

Como uma direção para trabalhos futuros, propõe-se a realização de novos experimentos de balanceamento de carga com o mecanismo MLB em um ambiente real, no qual espera-se fazer uso de *hardware switchs* compatíveis com o protocolo OpenFlow, além de utilizar tráfego de dados reais gerados por *hosts* reais.

Um segundo caminho a ser seguido, consiste em buscar a utilização da arquitetura SDN de forma distribuída com a utilização de mais de um controlador de rede simultaneamente. Dessa maneira, espera-se tornar à rede mais robusta e tolerante à falhas causadas por situações imprevisíveis, como por exemplo à queda da comunicação entre comutador e controlador de rede ocasionada por uma indisponibilidade no servidor que hospeda o controlador. Além disso, com o uso de mais controladores é possível aumentar a escalabilidade da rede com a inclusão de novos *hosts*. Desta maneira, será possível acompanhar o desempenho do mecanismo MLB em uma rede maior composta por um grande número de comutadores e consequentemente com comunicações entre origem e destino estabelecidas por intermédio de um número maior de saltos.

Como última ideia, sugere-se a realização de experimentos que testem à resiliência da rede quando submetida ao processo de balanceamento de carga com o mecanismo MLB. Experimentos simulando alterações não planejadas da topologia, como por exemplo a queda de um enlace do caminho em uso, pode ser um boa maneira de avaliar como o mecanismo irá gerenciar a comutação entre caminhos e a tarefa de manter atualizada a sua base de informações dos caminhos encontrados e ativos. Como também, pode ser verificado o desempenho do MLB frente a inclusão de novos nós e enlaces na topologia de rede formando novos caminhos para serem computados pelo mecanismo.

REFERÊNCIAS

AZZOUNI, A. et al. Limitations of openflow topology discovery protocol. In: *2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. [S.l.]: IEEE, 2017. p. 1–3. ISSN null.

BANFI, D. et al. Endpoint-transparent multipath transport with software-defined networks. In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. [S.l.: s.n.], 2016. p. 307–315. ISSN null.

BARBECHO, P. *New paradigms of legacy network features over SDN architecture*. 68 p. Dissertação (Mestrado) — Universidad Politécnica de Cataluna, Barcelona, 2017.

BHANDARKAR, S.; KHAN, K. A. Load balancing in software-defined network (SDN) based on traffic volume. *Advances in Computer Science and Information Technology (ACSIT)*, v. 2, n. 7, p. 72–76, april 2015.

BREDEL, M. et al. Flow-based load balancing in multipathed Layer-2 networks using openflow and multipath-TCP. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN'14), p. 213–214. ISBN 978-1-4503-2989-7.

BURIOL, L. S. *Roteamento do tráfego na internet: algoritmos para projeto e operação de redes com protocolo OSPF*. Tese (Doutorado) — Universidade Estadual de Campinas Faculdade de Engenharia Elétrica e de Computação, novembro 2003.

CORMEN, T. H. et al. *Algoritmos teoria e prática*. Rio de Janeiro: Elsevier, 2012. Tradução da 3ª edição americana. ISBN 978-85-352-3699-6.

COSTA, L. C. *Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais*. 108 p. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Juiz de Fora, 2016.

COSTA, L. H. M. K. et al. Grandes massas de dados na nuvem: Desafios e técnicas para inovação. In: SBC. *Minicurso apresentado no XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC*. [S.l.], 2012.

COUTO, R. de S. *Introdução às Redes Definidas por Software (SDN)*. 2019. Material de aula ministrado na Universidade Federal do Rio de Janeiro. Disponível em: ⟨https://www.gta.ufrj.br/~rodrigo/coe728.html⟩. Acesso em: 09 de dezembro de 2019.

COX, J. H. et al. Advancing software-defined networks: a survey. *IEEE Access*, IEEE, v. 5, p. 25487–25526, 2017. ISSN 2169-3536.

DUQUE, J.; BELTRáN, D.; LEGUIZAMóN, G. Opendaylight vs. floodlight: Comparative analysis of a load balancing algorithm for software defined networking. *International Journal of Communication Networks and Information Security*, v. 10, p. 348–357, january 2018.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: an intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833.

FOUNDATION, O. N. *OpenFlow switch specificationVersion 1.0*. 2009. Open Networking Foundation. Disponível em: ⟨https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf⟩. Acesso em: 09 de janeiro de 2020.

GANTZ, J.; REINSEL, D.; RYDNING, J. *The digitization of the world from edge to core*. 2018. Disponível em: ⟨https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf⟩. Acesso em: 09 de janeiro de 2020.

HAGBERG, A.; SCHULT, D.; SWART, P. *NetworkX - Algorithm all shortest path*. 2004. Biblioteca escrita em python para criação, manipulação e estudo de grafos e redes. Disponível em: ⟨https://networkx.github.io/documentation/networkx-2.3/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.all_shortest_paths.html#networkx.algorithms.shortest_paths.generic.all_shortest_paths⟩. Acesso em: 09 de dezembro de 2019.

HAGBERG, A.; SCHULT, D.; SWART, P. *NetworkX - Algorithm edje disjoint path*. 2004. Biblioteca escrita em python para criação, manipulação e estudo de grafos e redes. Disponível em: ⟨https://networkx.github.io/documentation/networkx-2.3/reference/algorithms/generated/networkx.algorithms.connectivity.disjoint_paths.edge_disjoint_paths.html#networkx.algorithms.connectivity.disjoint_paths.edge_disjoint_paths⟩. Acesso em: 09 de dezembro de 2019.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using NetworkX. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA: [s.n.], 2008. p. 11 – 15.

HASSAN, M. H. O. *Implementing Nayan Seth's dynamic load balancing algorithm in Software-Defined Networks: a case study*. 66 p. Dissertação (Mestrado) — Sudan University of Science and Technology, Sudan, 2017.

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à pesquisa operacional*. Trad. 8ª. São Paulo: McGraw-Hill /interamericana do Brasil Ltda., 2006. ISBN 85-868046-81.

Internet Systems Consortium. *Number of hosts advertised in the DNS*. 2019. Disponível em: ⟨https://downloads.isc.org/www/survey/reports/2018/01/⟩. Acesso em: 11 de setembro de 2019.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. 1ª. ed. [S.l.]: John Wiley & Sons, 1991. ISBN 978-0471503361.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, IEEE, v. 51, n. 2, p. 114–119, February 2013. ISSN 1558-1896.

KREUTZ, D. et al. Software-Defined Networking: a comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. Trad. 6ª. São Paulo: Pearson Education do Brasil Ltda., 2014. ISBN 978-85-8143-677-7.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: a survey. *IEEE Communications Surveys Tutorials*, IEEE, v. 16, n. 1, p. 493–512, First 2014. ISSN 1553-877X.

LONG, H. et al. *OSPF Traffic Engineering (OSPF-TE) link availability extension for links with variable discrete bandwidth*. RFC Editor, 2018. RFC 8330. (Request for Comments, 8330). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8330.txt⟩.

MALLIK, A.; HEGDE, S. A novel proposal to effectively combine multipath data forwarding for data center networks with congestion control and load balancing using Software-Defined Networking approach. In: *2014 International Conference on Recent Trends in Information Technology*. Chennai, India: IEEE, 2014. p. 1–7.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833.

MUDIGONDA, J. et al. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. San Jose, California: USENIX Association, 2010. (NSDI'10), p. 18.

MYERS, A.; NG, E.; ZHANG, H. Rethinking the service model: scaling ethernet to a million nodes. In: ACM SIGCOMM HOTNETS'04. San Diego, CA USA, 2004.

NGUYEN, T.-T.; KIM, D.-S. Accumulative-load aware routing in software-defined networks. In: *IEEE 13th International Conference on Industrial Informatics (INDIN)*. [S.l.: s.n.], 2015. p. 516–520. ISSN 1935-4576.

OPENDAYLIGHT. *OpenDaylight - The L2 Switch project provides Layer2 switch functionality*. 2017. Acesso em: 03 de dezembro de 2019. Disponível em: ⟨https://docs.opendaylight.org/en/stable-fluorine/user-guide/l2switch-user-guide.html⟩.

PERLMAN, R.; TOUCH, D. J. D. *Transparent Interconnection of Lots of Links (TRILL): problem and applicability statement*. RFC Editor, 2009. RFC 5556. (Request for Comments, 5556). Disponível em: ⟨https://rfc-editor.org/rfc/rfc5556.txt⟩.

PFAFF, B. et al. The design and implementation of open vswitch. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015. p. 117–130. ISBN 978-1-931971-218.

PRABHAVAT, S. et al. On load distribution over multipath networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 14, n. 3, p. 662–680, Third Quarter 2012. ISSN 1553-877X.

RAMDHANI, M. F.; HERTIANA, S. N.; DIRGANTARA, B. Multipath routing with load balancing and admission control in software-defined networking (SDN). In: *2016 4th International Conference on Information and Communication Technology (ICoICT)*. Bandung, Indonesia: IEEE, 2016. p. 1–6.

SCHARF, M.; FORD, A. *Multipath TCP (MPTCP) application interface considerations*. 2013. IRT.org. RFC 6897. Disponível em: ⟨https://www.irt.org/rfc/rfc6897.htm⟩. Acesso em: 26 de fevereiro de 2020.

SETH, N. *SDN Load balancing*. 2016. Algoritmo de balanceamento de carga codificado na linguagem de programação python. Disponível em: ⟨https://github.com/nayanseth/sdn-loadbalancing⟩. Acesso em: 16 de abril de 2019.

SILVA, P. H. D. da; JUNIOR, N. A. Ferramenta iperf: geração e medição de tráfego tcp e udp. *CBPF - Centro Brasileiro de Pesquisas Físicas*, v. 4, n. 2, p. 1–13, sep 2014.

SINGH, S. K.; DAS, T.; JUKAN, A. A survey on internet multipath routing and provisioning. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 4, p. 2157–2175, Fourthquarter 2015. ISSN 1553-877X.

STALLINGS, W. Software-defined networks and openflow. *The internet protocol Journal*, Chief Technology Office, Cisco Systems, Inc., v. 16, n. 1, p. 2–14, 2013.

SUN, Z. et al. An algorithm for the shortest pairs of arc-disjoint paths problem. In: *2012 8th International Conference on Natural Computation*. [S.l.: s.n.], 2012. p. 1001–1006. ISSN 2157-9563.

TSAI, J.; MOORS, T. A review of multipath routing protocols: from wireless ad hoc to mesh networks. In: *ACoRN early career researcher workshop on wireless multihop networking*. [S.l.: s.n.], 2006. v. 30.