UNIVERSIDADE FEDERAL FLUMINENSE

ALEX JOÃO BARBOSA DA SILVA

REDE AUTO-BALANCEADA DEFINIDA POR SOFTWARE EM CENTROS DE DADOS

NITERÓI

2015

UNIVERSIDADE FEDERAL FLUMINENSE

ALEX JOÃO BARBOSA DA SILVA

REDE AUTO-BALANCEADA DEFINIDA POR SOFTWARE EM CENTROS DE DADOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre. Área de concentração: Sistemas de Telecomunicações

Orientador:
NATALIA CASTRO FERNANDES

NITERÓI

2015

ALEX JOÃO BARBOSA DA SILVA

REDE AUTO-BALANCEADA DEFINIDA POR SOFTWARE EM CENTROS DE DADOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre. Área de concentração: Sistemas de Telecomunicações

Aprovada em Outubro de 2015.

BANCA EXAMINADORA

Profa. NATALIA CASTRO FERNANDES - Orientador, UFF

Prof. RICARDO CAMPANHA CARRANO, UFF

Prof. MICHAEL ANTHONY STANTON, RNP

Niterói

2015

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

S586 Silva, Alex João Barbosa da
Rede auto-balanceada definida por software em centros de dados / Alex João Barbosa da Silva. – Niterói, RJ : [s.n.], 2015.
99 f.

Dissertação (Mestrado em Engenharia de Telecomunicações) - Universidade Federal Fluminense, 2015.
Orientadores: Natalia Castro Fernandes.

1. Sistema de telecomunicações. 2. Rede de comunicação. 3. Balanceamento de fluxo. I. Título.

CDD 621.382

À minha família e aos meus amigos, por cada instante compartilhado nessa longa jornada.

Agradecimentos

Agradeço aos meus pais, João e Luza, pelo tempo e empenho dedicados à criação dos filhos, com especial atenção na educação pelo exemplo. Luta e perseverança para alcançar os objetivos são qualidades inestimáveis, transmitidas por eles de forma espontânea. Agradeço imensamente à minha irmã Elâine, sempre apoiando e incentivando na superação dos desafios que a vida nos apresenta. Aos amigos da F7 por compartilhar momentos com risadas garantidas e proporcionar o exercício da criatividade através da música. Agradeço aos amigos da Turma 1231 que, com descontração e um bom grau de curiosidade, tornaram o aprendizado de tecnologia uma atividade ainda mais prazerosa. Agradeço especialmente ao amigo Jack, que em uma conversa despretensiosa sobre projetos em andamento, trouxe a devida atenção para o mérito da taxa de uso como uma forma justa de quantificar o equilíbrio na utilização dos múltiplos caminhos disponíveis em uma rede. Aos demais amigos, cujos nomes são muitos para esse curto espaço, pela presença, troca de ideias e companheirismo. Aos meus colegas de trabalho, pela contribuição contínua no meu crescimento profissional e pessoal. Agradeço a professora Vera Caminha pela base em lógica de programação, ao professor Jorge Bitencourt pela ajuda para que essa jornada fosse iniciada e também ao professor José Brant, grande incentivador do meu ingresso no mestrado. Agradeço à Yona Lopes pelo apoio nas primeiras disciplinas do mestrado. Agradeço a minha orientadora Natalia Fernandes que me apresentou o tema Internet do Futuro e fomentou a pesquisa e desenvolvimento desse trabalho, fornecendo ferramentas e ensinamentos preciosos. Agradeço ainda aos colegas do Laboratório MidiaCom, pelo suporte fornecido a esse trabalho, especialmente ao Joacir, Cledson, Rafael, Diego, Rolim, Edelberto, Helga e à sempre prestativa Marister. Agradeço, por fim, aos professores participantes da banca examinadora pela disposição de seu tempo e conhecimento para analisar este trabalho.

Resumo

O advento do paradigma das Redes Definidas por Software - RDS (*Software-Defined Networking* - SDN) está promovendo a inovação e um desenvolvimento mais rápido das redes de computadores. Apesar do projeto de redes demandar abordagens robustas, mais capazes e flexíveis de forma a lidar com uma complexidade cada vez maior, a eficiência dos sistemas resultantes é, na maioria das vezes, limitada pela restrição de recursos em consequência de barreiras físicas, técnicas, financeiras, ou outros tipos de limitações. Desta forma, redes com enlaces redundantes têm sua capacidade potencial desperdiçada pelo uso de protocolos de roteamento distribuídos que não conseguem lidar de forma adequada com todos os caminhos disponíveis. Em especial, as redes dos centros de dados que possuem um grande número de enlaces redundantes e têm que fornecer a comunicação de forma eficiente entre seus equipamentos.

Nesta dissertação será apresentado o *Equalize*, um arcabouço criado segundo o paradigma RDS, que realiza o equilíbrio automático dos fluxos ingressantes na rede através dos enlaces disponíveis que estejam menos congestionados. A topologia da rede é "descoberta" de forma automática e a localização de um *host* é determinada pelo seu endereço de camada 2 (MAC), de forma que o mesmo pode migrar para qualquer ponto da rede mantendo seu endereço de camada 3 (IP) original. Para o cálculo do melhor caminho entre dois nós, o Equalize executa um algoritmo de Dijkstra adaptado, que emprega a utilização de cada enlace como parâmetro de custo. Assim, o Equalize estabelece na rede a funcionalidade de auto-balanceamento e também apresenta uma boa resiliência para falhas, além de suportar a migração de *hosts* (sejam eles físicos ou virtuais).

Com essas características, Equalize é bastante adequado para redes com facilidade para um gerenciamento centralizado e com múltiplos caminhos entre seus equipamentos, como as redes dos centros de dados.

Os resultados obtidos com a emulação de uma pequena rede mostram que a abordagem proposta é capaz de alcançar uma boa taxa de transferência na bissecção da rede, ao mesmo tempo em que garante o equilíbrio no uso dos enlaces contribuindo, assim, para o uso eficiente dos recursos de rede e melhorando a utilização de sua capacidade total disponível.

Palavras-chave: Balanceamento de fluxos, redes definidas por software, eficiência de rede.

Abstract

The advent of Software-Defined Networking (SDN) is promoting innovation and faster network development. Even though network design demands for robust, more capable and flexible approaches to cope with increasingly complexity, system efficiency is often limited due to resource bounding as a consequence of physical, technical, financial, or other types of constraints. Though, networks with multiple redundant links have their potential capacity wasted due to the use of distributed routing protocols that cannot adequately deal with all available paths. In particular, the data center networks that have a large number of redundant links and must provide efficient communications among edges.

In this dissertation we present Equalize, a framework built upon the SDN paradigm, that automatically balances the network incoming flows among the less congested available links. The network topology is automatically "discovered" and the location of a host is determined by its layer 2 (MAC) address, so that it can migrate to any point in the network while keeping its original layer 3 (IP) address. To calculate the best path between two nodes, Equalize runs an adapted Dijkstra's algorithm, employing each link utilization as the cost parameter. Thus, Equalize establishes the self-balancing functionality in the network and also presents a good resilience to failures, besides supporting the migration of hosts (whether physical or virtual).

Having those characteristics, Equalize is well suited for networks with ease of centralized management and multiple paths among edges, such as data center networks.

The results obtained by emulating a small network show that the proposed approach is able to achieve a good bisection throughput, while enforcing the balance in the use of links, thereby contributing to efficient use of network resources and improving the utilization of its total capacity available.

Keywords: Flow balancing, software-defined networking, network efficiency.

Lista de Figuras

Lista de Abreviaturas e Siglas

API *Application Programming Interface*

APSP *All Pairs Shortest Path*

ARP *Address Resolution Protocol*

ASIC *Application Specific Integrated Circuit*

CBE *Container-Based Emulation*

CLI *Command Line Interface*

CPU *Central Processing Unit*

DHFT *Dual-homed FatTree*

DIBS *Detour-Induced Buffer Sharing*

ECMP *Equal-Cost Multi-Path*

FIBRE *Future Internet Brazilian Environment for Experimentation*

GENI *Global Environment for Networking Innovation*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

ISP *Internet Service Provider*

ITU *International Telecommunication Union*

LAG *Link Aggregation Group*

LBC *Link Balancer Controller*

LLDP *Link Layer Discovery Protocol*

MAC *Media Access Control*

MPTCP *Multipath* TCP

MST *Minimum Spanning Tree*

NAT *Network Address Translation*

NS-3 *Network Simulator 3*

NSF *National Science Foundation*

OSI *Open Systems Interconnection*

OSPF *Open Shortest Path First*

RDS Redes Definidas por Software - *Software-Defined Networking* (SDN)

RNP Rede Nacional de Ensino e Pesquisa

RTT *Round Trip Time*

SPB *Shortest Path Bridging*

STP *Spanning Tree Protocol*

TCAM *Ternary Content-Addressable Memory*

TCP *Transport Control Protocol*

TRILL *Transparent Interconnection of Lots of Links*

TTL *Time to Live*

Sumário

Capítulo 1

Introdução

A Internet revolucionou a forma de viver das pessoas e já é considerada tão essencial para a sociedade moderna quanto as infraestruturas básicas de fornecimento de água, energia elétrica, etc. Esse grande sucesso é atribuído, dentre outros fatores, à simplicidade de sua arquitetura em camadas, que aplica o princípio fim-a-fim onde a funcionalidade é colocada nas camadas superiores (próxima da aplicação que a utiliza), e à interoperabilidade proporcionada por uma camada de cobertura capaz de suportar a interconexão de múltiplas tecnologias de rede como uma única inter rede lógica. Deste modo, a inteligência e maior complexidade são colocadas nos equipamentos das extremidades, enquanto os equipamentos no núcleo da rede realizam operações simples de armazenamento e encaminhamento de pacotes. Aplicações inovadoras são criadas com facilidade nas extremidades enquanto o núcleo da rede permanece simples e robusto, porém inflexível.

Não obstante, aplicações cada vez mais ricas e complexas tem surgido, e seus requisitos de desempenho e escalabilidade trazem uma variedade de desafios difíceis de serem abordados com a arquitetura atual da Internet [65]. Sendo a Internet composta por um vasto conjunto de redes diferentes que possuem em comum a utilização de certos protocolos e o fornecimento de certos serviços [80], é de se notar o empenho dos pesquisadores em torno desses assuntos. A arquitetura da Internet tem sido um tópico de pesquisa e exploração desde o seu início [49]. O imenso sucesso alcançado pela Internet, em conjunto com as limitações que ela apresenta para acomodar formas de uso e aplicações inovadoras, têm motivado vasta pesquisa tanto nas áreas afins quanto em novas áreas que podem vir a beneficiar-se dela. Nesse contexto, pesquisas abordando o tema Internet do Futuro têm buscado soluções aderentes à inovação, tendo como principal premissa a flexibilidade visando, assim, garantir a capacidade de mudar e evoluir [61].

O paradigma das Redes Definidas por Software - *Software-Defined Networking* (SDN)

(RDS) ainda está em sua infância, mesmo assim tem ajudado cada vez mais redes especializadas a evoluir de uma arquitetura ossificada como a da Internet e seus protocolos para sistemas flexíveis e programáveis. Das redes em campus de universidades a empresas, centros de dados, redes entre centros de dados e até pontos de troca de tráfego da Internet, propostas que alavancam as características da arquitetura RDS têm surgido em muitas formas e tamanhos [59, 56, 5, 6, 58, 92, 45, 79]. Consequentemente, a evolução das redes de computadores está ocorrendo a um ritmo mais rápido desde seu advento. No entanto, redes de computadores são complexas e difíceis de gerenciar [26] e muitos desafios ainda estão presentes.

1.1 Motivação

Redes de comunicações, em geral, possuem equipamentos e caminhos redundantes para garantir alta disponibilidade, porém, como na camada de enlace (camada 2) não existe um mecanismo como o *Time to Live* (TTL) para mitigar a ciclagem de pacotes, torna-se necessária a derivação da topologia da rede em uma subrede sem ciclos, realizada através do cálculo de uma árvore geradora (*spanning tree*) [70, 24] com o uso do protocolo *Spanning Tree Protocol* (STP) [41]. A topologia *lógica* resultante possui um único caminho - dentre todos os disponíveis - para a troca de dados entre dois equipamentos. Esta prática implica em requisitos de gerência mais simples, porém, por outro lado, acaba proporcionando um pior desempenho. Isso porque alguns enlaces acabam sobrecarregados, causando atraso na entrega dos pacotes e potencial descarte dos mesmos, enquanto caminhos redundantes estão livres. O impacto no desempenho é duplo: em primeiro lugar, devido ao desperdício de recursos de rede disponíveis e em segundo lugar, pela criação de gargalos que resultam em um congestionamento[1] crescente da rede.

O *Equal-Cost Multi-Path* (ECMP) [83, 40], foi concebido para amenizar esse problema, e faz uso de múltiplos caminhos de mesmo custo presentes na rede para distribuir os fluxos de pacotes. Nesse sentido, o ECMP contribui para aliviar o surgimento de congestionamento na rede e para um melhor balanceamento no uso dos enlaces disponíveis de forma a alcançar maior eficiência na utilização da rede. O protocolo *Open Shortest Path First* (OSPF) [62] (camada 3) foi um dos primeiros à introduzir o uso do ECMP.

[1] O congestionamento de rede ocorre quando, em um determinado instante de tempo, são demandados mais recursos do que os disponíveis na rede nesse mesmo instante de tempo. Ele é caracterizado por seus efeitos negativos, tais como o aumento no tempo de entrega dos pacotes, a diminuição na vazão útil (*goodput*) de enlaces e o desperdício de recursos que também decorrem do descarte de pacotes realizado por mecanismos de controle de congestionamento.

Recentemente, as novas propostas para encaminhamento na camada 2, *Transparent Interconnection of Lots of Links* (TRILL) [43, 71, 4] e *Shortest Path Bridging* (SPB) [42], também adotaram seu uso.

No entanto, para a distribuição dos fluxos, o ECMP realiza a escolha dos caminhos de forma estática, geralmente utilizando o resultado de uma função *hash*[2] sobre alguns campos do cabeçalho do pacote, o que pode levar a um balanceamento muito aquém do ideal se a diversidade de pacotes for baixa [83]. Ainda, uma solução na camada 3 apresenta a desvantagem de limitar a migração de máquinas virtuais à uma mesma subrede [63] e as propostas para aplicar o ECMP na camada 2 são muito recentes e não estão amplamente disponíveis em produtos comerciais [11]. Além disso, como o próprio nome sugere, o ECMP é limitado a caminhos de mesmo custo.

1.2 Objetivos

Uma alocação adaptativa dos fluxos, realizada com base na informação sobre o estado de utilização dos enlaces, tem potencial para permitir uma distribuição equilibrada e também contribuir para evitar o surgimento de congestionamento na rede. Buscando atingir esse objetivo de um uso mais eficiente da rede, nessa dissertação é proposto e implementado um arcabouço denominado Equalize, desenvolvido utilizando os princípios das RDS.

O Equalize foi implantado como um controlador de rede que monitora o uso dos enlaces e atua comandando o agendamento (*scheduling*) de novos fluxos. Esse agendamento dos fluxos é realizado através de qualquer um dos múltiplos caminhos disponíveis entre dois *hosts* (um *host* pode ser um computador ou um servidor), tendo como base a priorização dos enlaces que apresentam menor utilização durante o processo de tomada de decisão de encaminhamento dos pacotes. Em sua implementação, um algoritmo de Dijkstra é adaptado para aceitar como métrica de custo a razão de utilização de cada um dos enlaces da rede e realizar o cálculo dos menores caminhos (com menor utilização fim a fim) entre cada par de nós. Esses caminhos são, então, utilizados para o encaminhamento dos pacotes ao longo da rede, de forma que cada novo fluxo é direcionado pelo caminho de menor utilização fim a fim.

Testes foram conduzidos para avaliar o desempenho do encaminhamento realizado pelo Equalize em comparação com outras metodologias de roteamento, como a utilização

[2]Uma função *hash* é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Os valores retornados por uma função *hash* são chamados valores *hash*, códigos *hash*, somas *hash* (*hash sums*), *checksums* ou simplesmente *hashes* [85].

de caminho único (árvore geradora - *spanning tree*) e o ECMP.

1.3 Contribuições

Como principal contribuição desta dissertação, destaca-se a proposta apresentada de um mecanismo de atuação no encaminhamento de fluxos baseado em retro alimentação (*feedback*) pela coleta de medições de uso dos enlaces, que trabalha para a convergência no equilíbrio do tráfego transmitido através dos enlaces da rede. A hipótese abordada é que, ao encaminhar cada novo fluxo ingressante na rede pelo caminho menos congestionado, permite-se que os fluxos alcancem todo seu potencial maximizando a taxa de transferência da bissecção efetivamente utilizada assim como o equilíbrio no uso dos enlaces da rede.

1.4 Organização do trabalho

O restante desta dissertação está estruturado conforme explicação a seguir. O conceito de RDS é abordado no Capítulo 2. O Capítulo 3 apresenta os centros de dados e suas topologias de rede. Algumas técnicas de roteamento multi caminho são abordadas no Capítulo 4. No Capítulo 5, o conceito e a abordagem adotados para a concretização do encaminhamento equilibrado de fluxos são expostos, juntamente com os passos adotados para sua implementação, incluindo os componentes integrantes da solução proposta. O cenário de testes, a metodologia utilizada e os resultados obtidos são apresentados no Capítulo 6. Os trabalhos relacionados são abordados no Capítulo 7. Trabalhos futuros e as conclusões são apresentados no Capítulo 8.

Capítulo 2

RDS e OpenFlow

Neste capítulo é apresentada uma visão geral sobre os principais conceitos a respeito de Redes Definidas por Software - *Software-Defined Networking* (SDN) e OpenFlow, uma vez que esses são tópicos centrais dessa dissertação.

Uma rede de computadores ou rede de dados [90] é uma rede de telecomunicações que permite aos computadores trocar dados entre si. Nas redes de computadores, equipamentos de computação em rede (*networked computing devices*) transferem os dados através de um *enlace de dados* (*data link*). A rede de computadores mais conhecida é a Internet.

Equipamentos de computação em rede que originam, roteiam e terminam (consomem) os dados são chamados de *nós* da rede. Os nós podem incluir *hosts* tais como computadores pessoais, telefones, servidores e, de uma forma mais genérica, qualquer *hardware* de rede. As conexões entre os nós podem ser estabelecidas através de um meio cabeado ou sem fio. Dois nós podem ser ditos *ligados em rede* quando um nó é capaz de trocar informações com o outro, seja através de uma *conexão direta* ou *indireta*[1] entre si.

As redes de computadores diferem no meio de transmissão utilizado para transportar os seus sinais, nos protocolos de comunicação utilizados para organizar o seu tráfego, em seu tamanho, em sua topologia e em sua intenção organizacional.

Um grande número de aplicações é suportado pelas redes de computadores. Na maioria dos casos, os protocolos de comunicação específicos para cada aplicação são colocados em camadas superiores, isto é, são transportados como carga útil de outros protocolos de comunicação mais gerais.

O conjunto de protocolos da Internet é comumente conhecido como TCP/IP, porque

[1] Uma *conexão indireta* é uma conexão realizada através de nós intermediários.

os seus protocolos mais importantes, o *Transport Control Protocol* (TCP) e o *Internet Protocol* (IP) foram os primeiros protocolos de rede definidos neste padrão. Esse conjunto de protocolos pode ser visto como um modelo de camadas, onde cada camada é responsável por um grupo de tarefas, fornecendo um conjunto de serviços bem definidos para o protocolo da camada superior.

O TCP/IP fornece conectividade fim a fim, especificando como os dados devem ser empacotados, endereçados, transmitidos, encaminhados e recebidos no destino. Esta funcionalidade está organizada em quatro camadas de abstração que são usadas para classificar todos os protocolos relacionados, de acordo com o âmbito da comunicação em rede envolvida [13, 12]. As camadas são, da mais baixa para a mais alta: a camada de enlace (*link layer*), que contém métodos de comunicação para dados que permanecem dentro de um único segmento de rede (enlace); a camada de internet (*internet layer*), que conecta redes independentes, estabelecendo assim a comunicação entre redes (*internetworking*); a camada de transporte (*transport layer*), que manuseia a comunicação *host* a *host*; e a camada de aplicação (*application layer*), que fornece intercâmbio de dados de processo a processo para as aplicações.

A figura 2.1 mostra dois *hosts* conectados através de dois roteadores e as camadas correspondentes utilizadas em cada salto no caminho do fluxo de dados. A aplicação em cada *host* executa operações de ler e escrever como se os processos fossem diretamente ligados uns aos outros por algum tipo de tubo de dados. Todos os outros detalhes da comunicação permanecem ocultos a cada processo. Os mecanismos subjacentes que transmitem os dados entre os *hosts* estão localizados nas camadas inferiores de protocolos.

O modelo TCP/IP e muitos dos seus protocolos são mantidos pela *Internet Engineering Task Force* (IETF).

O modelo OSI (*Open Systems Interconnection model*) descreve um grupo fixo de sete camadas. As camadas são empilhadas na seguinte ordem [89]:

7. Camada de aplicação;
6. Camada de apresentação;
5. Camada de sessão;
4. Camada de transporte;
3. Camada de rede;
2. Camada de enlace de dados;

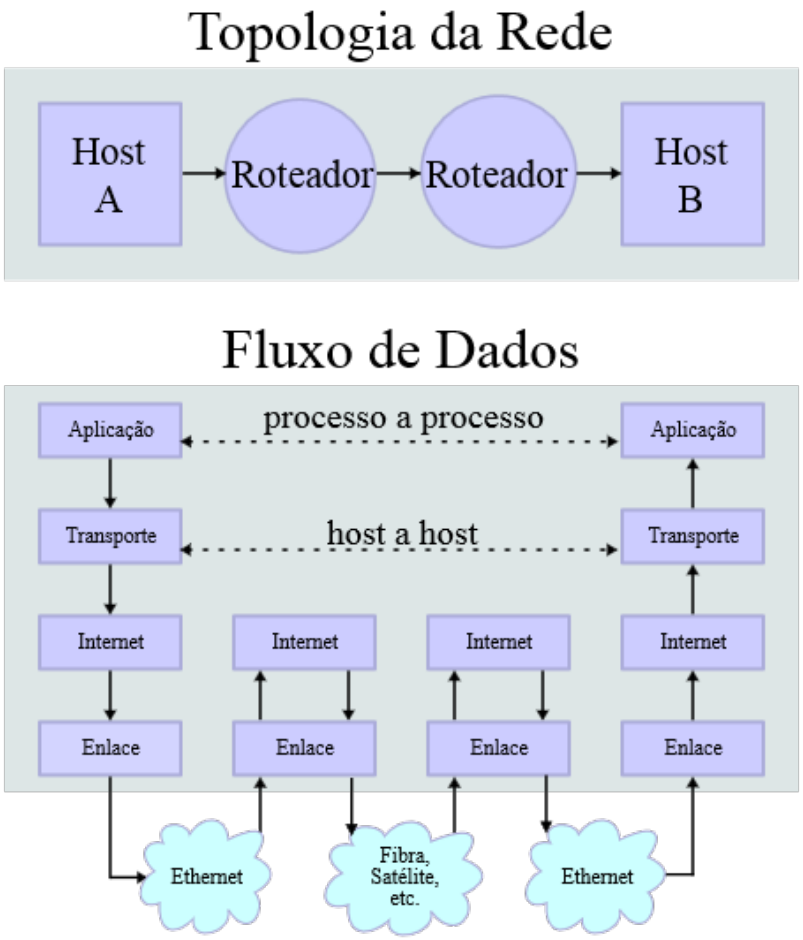

Figura 2.1: Exemplo de como a comunicação entre dois *hosts* flui através das camadas

1. Camada física.

Vários autores têm buscado incorporar as camadas 1 e 2 do modelo OSI ao modelo TCP/IP, uma vez que estas são comumente referidas nos padrões modernos (por exemplo, pelo IEEE e ITU). Isso comumente resulta em um modelo com cinco camadas, onde a camada de enlace é dividida nas camadas 1 e 2 do modelo OSI [87]. Surgiu, então, um modelo híbrido com 5 camadas [91]. São elas:

5. Camada de aplicação;

4. Camada de transporte;

3. Camada de rede;

2. Camada de enlace;

1. Camada física.

A respeito dos equipamentos de computação em rede, cabe destacar aqui a funcionalidade dos *comutadores* e dos *roteadores* em relação às camadas.

Um comutador de rede é um equipamentos capaz de encaminhar e de filtrar quadros de camada 2 entre suas portas, com base no endereço MAC de destino contido em cada quadro. Ele aprende e associa portas físicas a endereços MAC examinando os endereços de origem nos quadros recebidos. Se um endereço de destino em um quadro é desconhecido, o comutador transmite esse quadro para todas as suas portas (*broadcast*), exceto para a porta de origem.

Um roteador é um dispositivo de comunicação entre redes que realiza o encaminhamento de pacotes de acordo com as informações contidas no cabeçalho IP (camada 3). A informação de roteamento é, muitas vezes, processada em conjunto com a tabela de roteamento (ou tabela de encaminhamento). Um roteador usa sua tabela de roteamento para determinar para onde encaminhar os pacotes. Um destino numa tabela de encaminhamento pode incluir um interface "nula", também conhecida como interface de "buraco negro", porque os dados que chegam até ela não sofrem nenhum processamento adicional, isto é, nessa interface os pacotes são efetivamente descartados [90].

Em algumas redes, inclusive na Internet, a decisão sobre o encaminhamento dos dados é tomada por cada roteador individualmente com base no estado disseminado por outros roteadores da rede através de protocolos distribuídos. Cada roteador deve ser dotado de toda a "inteligência" necessária para reagir adequadamente. Qualquer mudança que se deseje aplicar ao comportamento adotado pelo processamento implica na alteração de configurações de forma individual em cada roteador da rede, para os casos mais simples e, alteração dos protocolos ou até mesmo criação de novos protocolos para os casos mais complexos. O esforço necessário no primeiro caso é proporcional à quantidade de parâmetros a serem alterados e ao número de elementos da rede. Para o segundo caso, torna-se uma missão hercúlea, tanto pela necessidade de se considerar o processamento distribuído, ou seja, o resultado final derivado da tomada de decisão individual por cada um dos elementos envolvidos, o que demanda testes em escala (desejavelmente em ambientes similares aos de produção), quanto pelo processo de padronização dos protocolos que pode levar anos considerando como premissa a garantia de interoperabilidade com os protocolos presentes. Além disso, tais roteadores são verdadeiras "caixas pretas", com pouca ou nenhuma abertura dos fabricantes para modificação do *software* dessas caixas, dificultando a inovação. Como alternativa para atender a demanda de novas aplicações, equipamentos intermediários (*middleboxes* como *firewalls*, NATs, balanceadores de carga,

etc.) têm sido criados e proliferam-se ao longo das redes [76, 65, 52].

Com o objetivo de solucionar esses problemas, a arquitetura RDS foi desenvolvida. A RDS é uma arquitetura de rede emergente, flexível e dinâmica, onde os *planos de dados* e *de controle* da rede são desacoplados, a inteligência e estado são logicamente centralizados no "controlador" onde as aplicações de rede residem, e a infraestrutura de rede subjacente é abstraída, garantindo assim programabilidade, automação e controle da rede [68]. O protocolo OpenFlow [59] é o mais utilizado na ligação entre os planos de controle e de dados. Através dele, o controlador se comunica com os elementos responsáveis pelo roteamento, adquirindo informações sobre o estado da rede e comandando ações para realização do comportamento adequado de roteamento.

Embora pareça que RDS surgiu de repente, muitas das ideias fundamentais já têm evoluído ao longo dos últimos vinte anos [26]. Em meados da década de 90, pesquisas na área de redes ativas propuseram tornar os elementos de rede mais flexíveis permitindo a programação de novas funcionalidades pelos usuários das redes [82, 81], porém, por questões práticas de desempenho e segurança, as arquiteturas propostas não ganharam adoção ampla [60] e a inovação em redes permaneceu restrita às abordagens tradicionais.

Em 2007, a arquitetura OpenFlow tomou forma buscando viabilizar inovação na pesquisa de redes através de uma infraestrutura flexível e programável. Para tal, herdou ideias de pesquisas e projetos que, cada um a seu modo, indicavam que os planos de dados e controle nos elementos de rede deveriam ser separados, e que as funções de controle e gerenciamento exercidas pelo plano de controle deveriam ser logicamente centralizadas de forma a superar a natureza fechada e pouco acessível dos elementos de rede atuais. Essa abordagem, no entanto, ainda era considerada por muitos como radical, pois o grande sucesso da Internet era, em parte, atribuído à simplicidade no coração da rede suportada pela pilha de protocolos já consolidada.

Ainda assim, o OpenFlow foi bem sucedido na implementação dessas ideias, ao propor uma interface bem definida entre o comutador (*switch*) e o controlador, além de um protocolo de comunicação padrão. Com isso, criou-se um modelo de plano de dados centrado na tabela de fluxos, a qual é "programada" pelo plano de controle através de um protocolo atuando como interface de aplicação (API). Além disso, a escolha inicial em ser compatível com o *hardware* de um comutador *ethernet* ou de um roteador permitiu aos fabricantes realizar uma simples atualização de *firmware* para habilitar as funcionalidades providas pelo OpenFlow, reduzindo o tempo para a disponibilização de produtos funcionais, em comparação com o tempo necessário para atualização ou até mesmo o desenvolvimento

de um novo *hardware*.

Em retrospectiva, alguns diferencias parecem ter contribuído para esse sucesso das redes OpenFlow dentro e fora do meio acadêmico:

- Nas redes tradicionais, existem muitos problemas de gerenciamento de rede, em especial nas redes de grande porte, o que leva administradores de rede a pressionarem por inovações e maior flexibilidade no núcleo da rede;
- A programação realizada através do plano de controle simplifica a adição de novas funcionalidades sem precisar atualizar todos os equipamentos das redes;
- O gerenciamento e a visibilidade da rede como um todo proporcionados pelo OpenFlow.

A arquitetura RDS expande os conceitos que deram origem ao OpenFlow para criar uma filosofia voltada para a evolução das redes de forma "programática". Isto é, de modo análogo ao que um sistema operacional é para comandar o *hardware* de um computador, a arquitetura RDS atribui ao plano de controle a missão de atuar como sistema operacional da rede. A partir de um ponto de vista intelectual, [26] apresenta a longa história de esforços para tornar as redes de computadores mais programáveis, e que terminou culminando no surgimento das RDS. Conceitos chave são destacados, além de esclarecer a relação entre as RDS e as tecnologias associadas, como virtualização, por exemplo. O amadurecimento do OpenFlow e das RDS também é abordado em [49], onde as experiências e lições aprendidas nos quatro ciclos de implantação em um período de três anos são compartilhadas. Muitas lições podem parecer óbvias, mas tiveram que ser aprendidas na prática para servir como base sólida para os avanços nos anos subsequentes. Um estudo abrangente sobre RDS é apresentado em [52].

Um equívoco frequentemente cometido é afirmar que RDS e OpenFlow são equivalentes. Na verdade, OpenFlow é apenas uma instância (amplamente popular) dos princípios das RDS, e diferentes APIs podem ser utilizadas no lugar do OpenFlow para controlar o comportamento de encaminhamento ao longo da rede [26].

Como o paradigma das RDS "herdou" muitos dos conceitos que estavam sendo difundidos junto com o OpenFlow, é relativamente comum encontrar na literatura contextos onde o termo OpenFlow refere-se a conceitos mais amplos de RDS (em oposição ao *protocolo* OpenFlow especificamente) e também contextos (geralmente de cunho histórico) onde o termo RDS refere-se à uma autêntica aplicação do OpenFlow.

A seguir, os principais conceitos das arquiteturas RDS e OpenFlow são apresentados.

2.1 Redes Definidas por Software - RDS

Os roteadores presentes nas redes convencionais incorporam uma forte integração entre os planos de controle e de dados. Este acoplamento tornou extremamente desafiadoras várias tarefas de gerenciamento de rede, tais como a depuração de problemas de configuração (*debugging*), e a previsão ou controle do comportamento de roteamento [26]. Alguns pesquisadores de rede buscaram uma abordagem alternativa de abertura do controle de rede, mais ou menos com base na analogia à relativa facilidade de se reprogramar um computador pessoal através da alteração do código fonte de seus programas.

As Redes Definidas por Software - *Software-Defined Networking* (SDN) proporcionam inovação na forma como as redes são projetadas e gerenciadas [26]. O paradigma das RDS alcança esse grande feito com: a separação do plano de controle do plano de dados; uma interface uniforme para comunicação entre os planos; a centralização lógica do plano de dados na forma de um sistema operacional de rede; e, eventualmente, uma interface padrão para a comunicação entre as aplicações de rede e o plano de controle.

O *plano de controle* decide como tratar o tráfego de dados e o *plano de dados* realiza o encaminhamento do tráfego de acordo com as decisões do plano de controle (que podem ser originadas nas aplicações de rede). Geralmente, esses dois planos estão integrados em cada elemento de rede, e cada elemento de rede trabalha de forma autônoma, culminando no funcionamento de forma distribuída das redes convencionais.

A Figura 2.2 mostra as camadas da arquitetura das Redes Definidas por Software - *Software-Defined Networking* (SDN). Os componentes da arquitetura RDS são:

1. Elementos de encaminhamento (comutadores). Estes elementos fazem parte da camada de Infraestrutura;
2. Entidade de controle logicamente centralizada (controlador). O controlador (ou *software* de controle) está na camada de controle;
3. Aplicações de rede. Integram a camada de aplicação;
4. Interface de comunicação entre o controlador e os elementos de encaminhamento. Também conhecida como *southbound*;

5. Interface de comunicação entre o controlador e as aplicações de rede. Também conhecida como *northbound*.

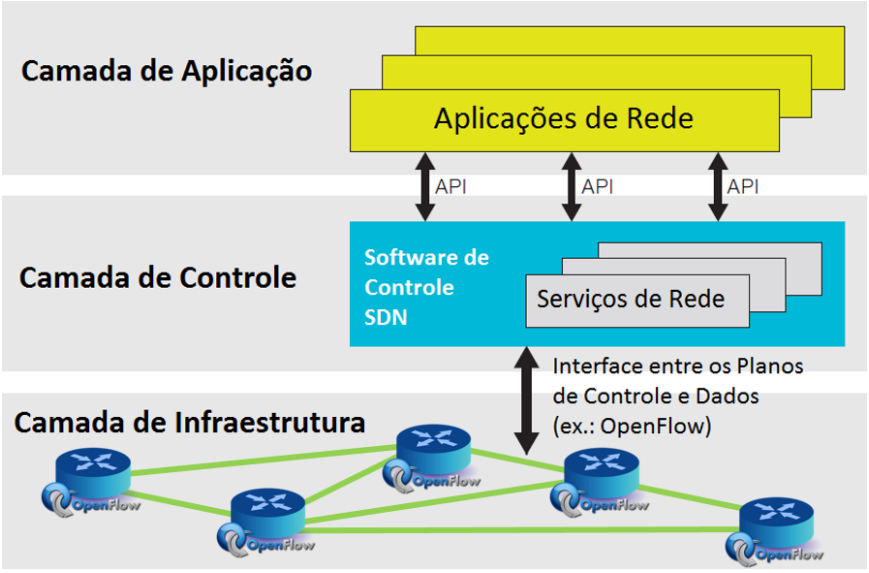

Figura 2.2: Arquitetura da Rede Definida por Software

Os benefícios propostos pelo paradigma das RDS variam desde oferecer controle centralizado, algoritmos simplificados, redução dos preços do *hardware* de rede, eliminação de *middleboxes*, até a possibilidade de criação e implantação de aplicações de rede por terceiros [65].

Um aspecto importante das RDS é a ligação entre o plano de dados e o plano de controle (*southbound*). Como os elementos de encaminhamento são controlados por uma interface aberta, é importante que esta ligação permaneça disponível e segura.

Enquanto a interação controlador-comutador (*southbound*) é bem definida através de uma interface de comunicação padronizada, ainda não há um consenso sobre uma forma comum de interação entre o controlador e as aplicações (ou serviços) de rede (*northbound*). Assim, tal interação tem sido implementada de forma pontual para cada aplicação em particular.

Embora RDS seja frequentemente anunciada como a solução para todos os problemas de rede, vale lembrar que RDS é mais uma (grande) ferramenta para resolver os problemas de rede com maior facilidade. A RDS meramente coloca o poder nas mãos de seus usuários para desenvolver novas aplicações e soluções para problemas de rede. Manter a visão ousada da RDS implica em continuar pensando "fora da caixa" sobre as melhores maneiras de se programar a rede, sem se restringir às limitações impostas pelas tecnologias atuais [26].

2.2 OpenFlow

Do ponto de vista atual, a abordagem apresentada pelo OpenFlow utiliza os conceitos e princípios da arquitetura RDS, e o *protocolo* OpenFlow [2] é a primeira interface de comunicação padrão definida entre os planos de controle e de dados de uma arquitetura RDS. O OpenFlow é um protocolo aberto que permite o manuseio e o acesso direto ao plano de dados (tabela de fluxos - *flow table*) dos equipamentos de rede como comutadores, tanto físicos quanto virtuais [59] de modelos e fabricantes diversos.

O plano de controle atua diretamente sobre o estado dos elementos de rede através de uma API bem definida e o OpenFlow é um exemplo proeminente de tal API.

A Figura 2.3 mostra um elemento de comutação OpenFlow comunicando-se com o controlador através de um canal seguro implementado em *software*. A tabela de fluxos implementada em *hardware* realiza o encaminhamento dos fluxos que chegam dos *hosts* conectados, de acordo com as instruções "instaladas" pelo controlador.

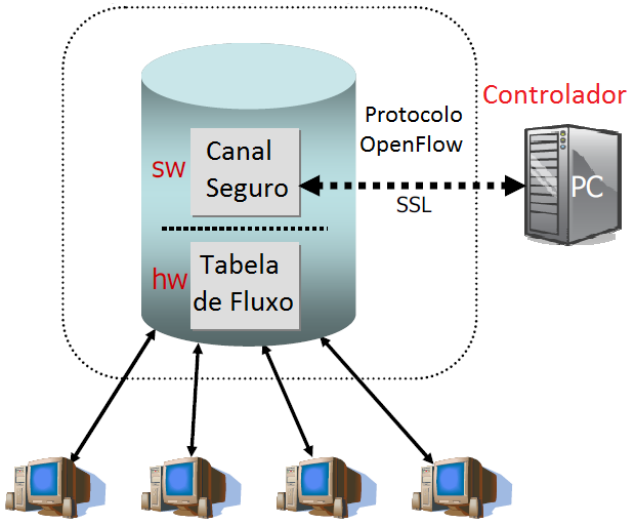

Figura 2.3: Arquitetura OpenFlow [59]

O OpenFlow [59] separa o plano de controle do plano de dados nos comutadores, permitindo que uma entidade controladora separada possa modificar as regras de encaminhamento, oferecendo assim uma grande flexibilidade para o roteamento dos fluxos de dados de uma rede, além da liberdade para modificar o comportamento de parte da rede, deixando o restante intacto. Ele foi proposto inicialmente como uma forma de possibilitar que pesquisadores conduzissem experimentos nas redes em produção. Para tanto, as redes seriam virtualizadas, permitindo que os equipamentos tratassem tanto o tráfego de produção quanto o tráfego experimental.

O protocolo OpenFlow [2, 66, 67] padroniza a troca de informações entre os planos de controle e de dados, e é a interface aberta que faz a conexão entre eles, levando os comandos e requisições do controlador para o comutador e as notificações e respostas do comutador para o controlador.

Em sua versão 1.0, a especificação do protocolo OpenFlow estabelece que o comutador deve possuir pelo menos uma tabela de fluxos (*flow table*). A tabela de fluxos é responsável pelo armazenamento das entradas de fluxo (*flow entries*), onde uma entrada de fluxo determina como os pacotes de um fluxo serão processados e encaminhados. Para oferecer um desempenho superior, a tabela de fluxos (ou parte dela) é implementada em *hardware*, podendo até mesmo existir mais de uma tabela em um mesmo elemento de encaminhamento, com o objetivo de realizar buscas e operações mais específicas. Para suportar buscas com campos coringa, são usadas memórias ternárias de conteúdo endereçável (*Ternary Content-Addressable Memory* (TCAM)).

Cada entrada de fluxo é composta por três componentes:

1. Um conjunto composto por 12 campos, sendo 11 com informações presentes no cabeçalho de um pacote mais um campo indicando a porta de ingresso no comutador, utilizado para comparação com os campos do cabeçalho dos pacotes ingressantes no comutador;

2. Uma lista de ações que ditam o que fazer com os pacotes que correspondem à comparação realizada;

3. Uma coleção de medições (estatísticas) de um fluxo, com o número de *bytes*, número de pacotes e o tempo decorrido desde a última vez que um pacote ingressante correspondeu a esse fluxo.

Quando um pacote chega a um comutador OpenFlow, a informação de seu cabeçalho é processada e então comparada com as entradas na tabela de fluxos. Essa comparação se inicia na primeira tabela de fluxos e pode continuar pelas tabelas de fluxos subsequentes. Se uma correspondência for encontrada, o comutador aplica as ações apropriadas ao pacote e atualiza as medições para a entrada de fluxo correspondente na tabela. Se nenhuma correspondência for encontrada, na versão 1.0 do protocolo, a ação realizada pelo comutador será o encaminhado direto do pacote para o controlador. Caso uma versão mais recente do protocolo esteja em uso, a ação tomada dependerá das instruções contidas em uma entrada de fluxo especial denominada *table-miss*. Ao receber esse pacote do comutador, o controlador irá, então, determinar como o pacote deverá ser tratado.

O controlador remoto pode, através do protocolo OpenFlow, solicitar medições ao comutador e também comandar a adição, atualização ou remoção de entradas de fluxo nas tabelas de fluxo do comutador. O envio de comandos pode ocorrer tanto de forma *reativa* (em resposta à chegada de um pacote), conforme a sequência acima, quanto de forma *proativa*.

As entradas de fluxo OpenFlow podem definir o comportamento de encaminhamento dos fluxos com base em qualquer combinação dentre doze campos de comparação. Assim, o OpenFlow oferece flexilidade para definir os fluxos de dados da rede, permitindo a um único elemento de encaminhamento desempenhar papéis que em redes tradicionais são exercidos por elementos distintos como roteadores e *firewalls*.

O antecessor imediato do OpenFlow foi o projeto SANE / Ethane [14] que, em 2006, definiu uma nova arquitetura para redes corporativas. O foco de Ethane estava no uso de um controlador centralizado para gerir as políticas e a segurança em uma rede, fornecendo controle de acesso baseado em identidade. [65]

O trabalho na implantação de sistemas OpenFlow levou à noção de um sistema operacional de rede [31]. Um sistema operacional de rede é um software que abstrai a lógica e os detalhes intrínsecos à instalação de estado nos comutadores de rede, simplificando o trabalho das aplicações que controlam o comportamento da rede. Em termos mais gerais, o surgimento de um sistema operacional de rede ofereceu uma decomposição conceitual do funcionamento da rede em três partes [51]:

1. Um plano de dados com uma interface aberta;
2. Uma camada de gerenciamento que é responsável por manter uma visão consistente do estado da rede;
3. A lógica de controle que executa várias operações em função da sua visão do estado da rede.

2.3 Elementos de encaminhamento

A estrutura física da rede pode envolver diferentes equipamentos, como por exemplo comutadores, roteadores, pontos de acesso sem fio, etc. Como em RDS os algoritmos de encaminhamento/roteamento e a lógica de controle ficam a cargo do controlador de rede, os elementos de encaminhamento são comumente representados como *hardware básico de*

encaminhamento, acessível por uma interface de controle aberta (não proprietária). Desta forma, na terminologia RDS, tais elementos são comumente referidos simplesmente como "comutadores".

Em relação aos roteadores IP tradicionais, os elementos de encaminhamento RDS suportam a comparação de um número maior de campos de cabeçalho, e consequentemente comparações mais flexíveis. Logo, o número de ações possíveis para execução em resposta à chegada de um pacote também é maior, o que por um lado torna o equipamento RDS multifuncional, mas por outro, torna as regras na tabela de fluxos OpenFlow mais complexas que as regras de encaminhamento em roteadores IP tradicionais [65].

Para alcançar seu potencial, comutadores que implementam uma arquitetura RDS baseada em fluxos, como o OpenFlow, podem necessitar de entradas de tabela de fluxos adicionais, maior espaço de *buffer* e contadores extras que não são fáceis de se implementar nos ASICs dos comutadores tradicionais, por estes terem sido desenvolvidos para plataformas proprietárias onde os planos de dados e controle estão integrados e oferecem acesso a uma gama limitada de parâmetros configuráveis.

A versão 1.0 do OpenFlow foi a primeira a ter suporte oficial de fabricantes que, no entanto, simplesmente adaptaram o *software* de alguns produtos já existentes para apresentar ao controlador de rede a interface padronizada. Desta forma, a funcionalidade e flexibilidade de configuração apresentadas por tais equipamentos tiveram ganhos significativos, apesar do desempenho geral do processamento das regras de encaminhamento ter sido impactado pelo *hardware* subjacente, em comparação com o desempenho apresentado pelo equipamento cumprindo apenas sua função original. Tal impacto ocorreu principalmente porque o processador desses equipamentos passou a exercer uma atividade extra, atuando diretamente na comunicação entre o controlador e o plano de dados. Outra limitação se deu na quantidade de regras de encaminhamento, pois o tamanho da memória permaneceu o mesmo, apesar das regras OpenFlow demandarem maior espaço.

Entretanto, já surgiram comutadores puramente OpenFlow, com desempenho superior aos equipamentos híbridos.

2.4 O controlador de rede

Na arquitetura RDS, o plano de controle logicamente centralizado é constituído por um controlador de rede (ou sistema operacional de rede), que constrói e apresenta um mapa lógico de toda a rede para serviços ou aplicações de controle de rede situados na camada

superior, e possui as funcionalidades de segmentar e virtualizar a rede subjacente. A arquitetura RDS consolida o plano de controle, de modo que um único *software* de controle (o controlador de rede) controla vários elementos do plano de dados.

Assim, o controlador de rede é responsável por manusear a distribuição de estado ao longo da rede, através da coleta de informação dos comutadores, e do processamento e configuração do estado de controle adequado nos comutadores.

Um controlador centralizado *fisicamente* representa um único ponto de falha para a rede como um todo, por esse motivo, o OpenFlow permite a conexão de vários controladores a um comutador, o que possibilita que o(s) controlador(es) de reserva possa(m) assumir o controle no caso de uma falha no controlador principal ou no meio de comunicação com o comutador.

No caso de um controlador distribuído em vários servidores físicos, o *software* de controle também é responsável pela coordenação do estado entre os servidores.

O controlador de rede provê uma interface programática na qual desenvolvedores podem criar um ampla variedade de aplicações de gerenciamento de rede, possibilitando a disponibilização das funcionalidades de roteamento e controle de acesso, por exemplo, pela implementação da lógica de controle necessária para essas funcionalidades.

A configuração do estado de controle nos comutadores pode ocorrer de duas formas: reativa e proativa.

2.4.1 Controle reativo vs. proativo

O plano de controle pode ser *reativo* ao manusear novos fluxos somente após sua chegada, ou *proativo*, inserindo entradas de fluxo, antes mesmo do fluxo chegar, ou pode ainda usar uma combinação dessas duas abordagens com base em fluxos específicos. [49]

Sob o modelo de controle reativo, os elementos de encaminhamento devem consultar um controlador a cada vez que uma nova decisão precisa ser tomada, por exemplo, quando um pacote de um novo fluxo chega a um comutador. No caso do controle com base na granularidade por fluxo, haverá uma pequena redução no desempenho devido ao atraso incorrido pelo primeiro pacote de cada novo fluxo ao ser enviado para o controlador, que irá decidir a ação para o novo fluxo (por exemplo, encaminhar ou descartar) e configurar o comutador para refletir essa ação. Após essa etapa, os futuros pacotes pertencentes a esse fluxo serão transportados à taxa de linha (capacidade nominal da porta) no *hardware* de

encaminhamento, graças a realização da comparação com a regra já instalada diretamente na tabela de fluxos. Como o controlador é envolvido na tomada de uma quantidade maior de decisões a respeito dos novos fluxos que surgem na rede, ele também adquire uma maior visibilidade sobre tais fluxos.

Enquanto o atraso sofrido pelo primeiro pacote pode ser insignificante em muitos casos, ele pode tornar-se uma preocupação se o controlador estiver geograficamente distante, devido à latência extra inserida entre o comutador e o controlador, ou se a maioria dos fluxos é de curta duração (como fluxos de um único pacote). Há também a possibilidade de alguns problemas de escalabilidade em redes maiores, pois o controlador deve ser capaz de lidar com um volume maior de novos pedidos de fluxo.

No método de controle proativo, o controlador instala antecipadamente as regras de fluxo nos comutadores, ou seja, antes mesmo de um pacote com as características do fluxo determinado chegar a um dos comutadores e esse, por sua vez, enviá-lo ao controlador devido a não coincidência dsse fluxo com alguma regra em sua tabela de fluxos. Após a instalação das regras na tabela de fluxos do comutador, qualquer novo pacote pertencente aos fluxos envolvidos será diretamente transportado à taxa de linha no *hardware* de encaminhamento. Nesse caso, o controlador só terá conhecimento a respeito do estado desses fluxos caso solicite explicitamente essa informação aos comutadores.

Assim, essas duas formas de controle implicam em uma relação de compromisso entre desempenho e visibilidade. O modelo de controle reativo impacta no desempenho máximo alcançável ao demandar que o primeiro pacote de cada novo fluxo seja enviado ao controlador, mas fornece ao controlador uma boa visibilidade sobre os fluxos presentes na rede. Já o modelo de controle proativo permite o máximo desempenho das interfaces do comutador ao deixá-lo configurado antecipadamente para tratar um determinado fluxo, mas limita a visibilidade do controlador sobre a presença efetiva desse fluxo na rede.

2.5 Considerações sobre desempenho nas Redes Definidas por Software (RDS)

Se por um lado a arquitetura RDS traz muitas vantagens importantes, decorrentes de seus atributos, por outro ela levanta desafios de escalabilidade, desempenho, robustez e segurança.

A separação realizada entre o plano de dados e o plano de controle sugere uma penalidade de desempenho em termos de atraso adicional para as operações de controle

(especialmente para a instalação de um fluxo), descoberta de topologia, e recuperação de falhas. Um plano de controle centralizado (logicamente) sugere um desafio de escalabilidade e traz preocupações como a possível existência de um ponto único de falha. A presença de limitações de *hardware* no comutador, como o tamanho máximo da tabela de fluxos e a capacidade da CPU, podem ser um problema para uma implantação de produção (não experimental) [49].

O tempo de instalação de fluxo é uma métrica importante de desempenho para Redes Definidas por Software - *Software-Defined Networking* (SDN). O subsistema de CPU de um comutador, responsável pelo tráfego de controle, é o fator determinante nesse quesito. Depende dos fornecedores renovar os produtos com subsistemas de CPU de alto desempenho. O número de entradas de tabela de fluxo suportado pela maioria dos comutadores comerciais é uma limitação no curto prazo. Futuros comutadores OpenFlow terão de suportar um número muito maior de entradas de tabela de fluxo no *hardware* para alcançar um melhor desempenho e uma maior escala [49]. Portanto, o paradigma das RDS exigirá pesquisa continuada e mais implementações diversas para alcançar o seu pleno potencial.

2.6 Ferramentas para desenvolvimento e análise de Redes Definidas por Software (RDS)

A arquitetura RDS foi proposta para facilitar a evolução da rede e a inovação ao permitir a rápida implantação de novos serviços e protocolos. Nesta seção, é apresentada uma visão geral sobre as ferramentas e ambientes atualmente disponíveis para o desenvolvimento de serviços e aplicações baseados em RDS. Muitas ferramentas para desenvolvimento e análise de RDS estão disponíveis, tanto gratuitas quanto comerciais, de código aberto e proprietário, nas áreas de comutadores por *software* (*software switches*), plataformas de controle, ferramentas para verificação de código e *debugging*, simuladores, emuladores e *testbeds*.

Mais detalhes sobre os controladores utilizados neste trabalho encontram-se na Seção 2.6.2.

2.6.1 Comutadores por *software* (*software switches*)

Um comutador desenvolvido em *software* fornece ao sistema operacional hospedeiro a funcionalidade de comutação de pacotes encontrada nos comutadores por *hardware*. Em

ambientes virtualizados, instâncias de comutadores por *software* podem formar uma rede virtualizada.

Em conjunto com a criação da especificação do protocolo OpenFlow 1.0 foi desenvolvido um comutador por *software* de referência. Outro comutador de uso geral desenvolvido em *software*, mas que também suporta o protocolo OpenFlow é o Open vSwitch.

2.6.1.1 Comutador por *software* de referência OpenFlow

O comutador por *software* de referência OpenFlow [3] foi desenvolvido para o sistema operacional Linux e possibilita a comutação de pacotes entre múltiplas interfaces de rede, de acordo com as configurações de fluxo aplicadas pelo controlador de rede. Essa é uma implementação de referência OpenFlow que acompanha a especificação.

2.6.1.2 Open vSwitch

O Open vSwitch [73] é um comutador virtual multi camada de código aberto com qualidade de produção, que suporta diversas interfaces e protocolos padronizados, além de também ter sido projetado para permitir automação massiva das configurações de rede através de extensão programática.

Nos experimentos realizados nessa dissertação, o Open vSwitch é utilizado como um comutador OpenFlow remotamente controlado.

2.6.2 Plataformas de controle

Uma plataforma de controle RDS normalmente contém uma coleção de módulos "acopláveis" que podem executar diferentes tarefas de rede. Algumas das tarefas básicas incluem a criação de um inventário com os equipamentos que fazem parte da rede e seus recursos, e também a coleta e o processamento de medições. Podem ser inseridas extensões que melhoram a funcionalidade e suportam tarefas mais avançadas, como por exemplo a execução de algoritmos que realizam análises e orquestram novas regras em toda a rede.

Existem diversos controladores RDS disponíveis. A seguir, são descritos os controladores relacionados a este trabalho.

2.6.2.1 Controlador Beacon

O Beacon [25] é um controlador OpenFlow de código aberto baseado em Java que tem sido amplamente utilizado para educação e pesquisa. Foi criado com os objetivos de:

1. Melhorar a produtividade dos desenvolvedores de *software* para redes;
2. Fornecer a possibilidade de iniciar e de parar as aplicações existentes e novas em tempo de execução;
3. Possuir alto desempenho.

A Figura 2.4 mostra a estrutura do controlador Beacon.

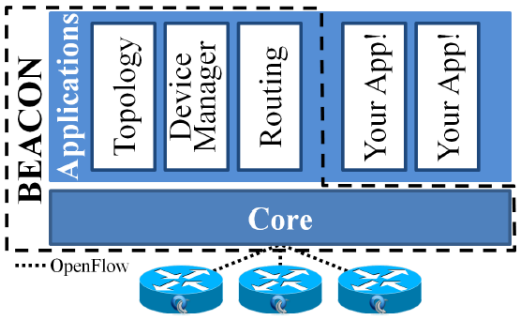

Figura 2.4: Estrutura do controlador Beacon

2.6.2.2 Controlador RipL-POX

O Ripcord-Lite for POX (RipL-POX) [36] é um controlador OpenFlow criado sobre o RipL [37, 38] para centros de dados e escrito na linguagem Python. Ele utiliza uma descrição estática da topologia da rede para calcular e configurar os caminhos entre dois nós disponíveis na rede.

A configuração pode ocorrer de duas formas, proativa ou reativa. Na configuração proativa, todos os caminhos possíveis são configurados logo que todos os comutadores da rede tenham se conectado ao controlador. Na configuração reativa, são configurados caminhos completos para cada novo fluxo que entra na rede.

A forma pela qual o cálculo dos caminhos (roteamento) é realizado pode ser previamente escolhida entre *árvore geradora*, *aleatória* ou *função hash*. A *função hash* utilizada depende da forma de configuração escolhida, sendo baseada nos campos de cabeçalho da camada 2 (endereços MAC de origem e de destino) para a configuração proativa, e nos

campos de cabeçalho das camadas 3 e 4 (IPs de origem e de destino, protocolo L4, portas de origem e de destino) para a configuração reativa.

2.6.3 FlowVisor

O FlowVisor [77] é um controlador OpenFlow de propósito especial que age como um *proxy* transparente entre comutadores OpenFlow e vários controladores OpenFlow, permitindo que um mesmo elemento de encaminhamento possa ser compartilhado entre múltiplas redes lógicas, cada qual com sua própria lógica de encaminhamento.

2.6.4 Simuladores, emuladores e *testbeds*

2.6.4.1 NS-3

O NS-3 (*Network Simulator 3*) [64] é um simulador de redes de eventos discretos, voltado principalmente para pesquisa e uso educacional. Ele possui suporte à simulação de redes OpenFlow e permite avaliar o funcionamento da rede em termos de tráfego e desempenho [17].

O elemento comutador OpenFlow do ns-3 modela um comutador com capacidade OpenFlow. Ele foi projetado para expressar o uso básico do protocolo OpenFlow, com a manutenção de uma tabela de fluxos e TCAM virtuais para fornecer resultados similares aos apresentados por equipamentos OpenFlow reais.

Para ser "instalado" nos nós da rede simulada, o módulo OpenFlow do NS-3 apresenta as classes *OpenFlowSwitchNetDevice* e *OpenFlowSwitchHelper*. Ele também usa um conjunto de *NetDevices* para serem configurados como portas, e age como o intermediário entre elas, recebendo um pacote em uma porta e encaminhando-o para outra, ou até para todas as portas (exceto a que recebeu o pacote inicialmente) quando uma inundação (*flooding*) é realizada.

Como um comutador OpenFlow, o módulo OpenFlow mantém uma tabela de fluxos configurável que pode comparar pacotes por seus cabeçalhos e executar ações diferentes com o pacote de acordo com a correspondência encontrada na tabela de fluxos. O entendimento das mensagens de configuração OpenFlow que módulo possui permite que estas mantenham o mesmo formato utilizado por comutadores reais compatíveis com o OpenFlow. Desta forma, usuários testando um controlador com o NS-3 não precisarão reescrever o código para que ele funcione controlando comutadores reais compatíveis com

o OpenFlow.

2.6.4.2 Mininet

O Mininet [54, 33, 53] permite a emulação de uma rede OpenFlow inteira em uma única máquina, simplificando o processo inicial de desenvolvimento e implantação. Novos serviços, aplicações e protocolos podem ser primeiramente desenvolvidos e testados em uma emulação do ambiente de implantação planejado, antes de passar para o *hardware* real.

O Mininet foi desenvolvido para ajudar os pesquisadores e desenvolvedores a emular rapidamente as RDS com topologia, escala, e matrizes de tráfego arbitrárias. Nesta rede emulada, é possível desenvolver e experimentar com uma verdadeira pilha de controle RDS que pode, posteriormente, ser implantada em uma infraestrutura física sem quaisquer alterações.

O Mininet aplica a técnica de Emulação Baseada em Contêiner (*Container-Based Emulation* (CBE)) [69], uma forma de virtualização a nível de processo onde muitos aspectos do sistema são compartilhados e que é mais leve que a virtualização de um sistema operacional inteiro. A CBE busca oferecer um ambiente o mais próximo possível do fornecido por uma máquina virtual, mas sem o custo extra decorrente da execução de um *kernel* separado e da simulação de todo o *hardware* envolvido. Isto é alcançado através da combinação de funcionalidades de segurança presentes no *kernel*, tais como espaços de nomes (*namespaces*), controle mandatório de acesso e grupos de controle [69]. Ao compartilhar recursos do sistema operacional, como tabelas de paginação, estruturas de dados no *kernel*, e sistema de arquivos, um contêiner a nível de sistema operacional alcança maior escalabilidade do que sistemas baseados em máquinas virtuais completas, permitindo assim, centenas de pequenos *hosts* e comutadores virtuais em um único sistema operacional.

Pela combinação de virtualização leve com uma Interface de Linha de Comando (CLI) e uma Interface de Programação de Aplicação (API) extensíveis, o Mininet oferece um fluxo de trabalho de prototipagem rápida para a criação, interação, customização e compartilhamento de uma rede OpenFlow, assim como um caminho suave para a execução em *hardware* real. Desta forma, o Mininet oferece um ambiente onde os *hosts*, comutadores e enlaces integrantes da rede são instanciados rapidamente, de acordo com uma topologia pré-estabelecida.

2.6.4.3 *Testbeds*

Os *Testbeds* para pesquisas em Internet do Futuro são formados por redes programáveis dedicadas à experimentação, conectadas à Internet atual. A utilização de *testbeds* é necessária para testar e validar as novas arquiteturas de rede, sem perturbar o funcionamento da Internet corrente, e permite aos pesquisadores (ou "experimentadores") avaliar novos modelos de arquiteturas de rede e aplicações.

Dentre outros *testbeds* existentes, dois são destacados aqui com grande potencial para uso em experimentos futuros relativos à proposta apresentada nesse trabalho.

O *testbed Global Environment for Networking Innovation* (GENI) [10] é um laboratório virtual distribuído, implantado nos Estados Unidos para a realização de experimentos transformadores em escala nas áreas de redes, serviços e segurança. Projetado em resposta a preocupações sobre a ossificação da Internet, o GENI possibilita uma ampla gama de experimentos em uma variedade de áreas, incluindo redes totalmente projetadas do zero (*clean-slate*), projeto e avaliação de protocolos, oferta de serviços distribuídos, integração de redes sociais, gerenciamento de conteúdo e implantação de serviços na própria rede. Ele é suportado pela *National Science Foundation* (NSF), e está disponível sem custo para a pesquisa e o uso em sala de aula.

O *testbed Future Internet Brazilian Environment for Experimentation* (FIBRE) [27] é uma estrutura para pesquisa construída no âmbito de um projeto financiado pela "*2010 Brazil-EU Coordinated Call in ICT*".

Atualmente, a infraestrutura do FIBRE é constituída por uma federação de 11 *testbeds* locais, também chamados de "nós de experimentação" ou simplesmente "ilhas". Cada ilha possui um conjunto de equipamentos de rede para suportar experiências em ambas as tecnologias, fixa e sem fios, com elementos que integram o padrão OpenFlow. Essas ilhas são conectadas através uma rede sobreposta (*overlay*) ao *backbone* da Rede Nacional de Ensino e Pesquisa (RNP), composta por duas redes separadas: uma para o plano de controle e outra para o plano de dados dos experimentos. Esta rede sobreposta é denominada "FIBREnet".

Originalmente chamado de "*Future Internet testbeds / experimentation between BRazil and Europe*" (FIBRE), o projeto foi lançado em outubro de 2011 com o objetivo de implantar uma nova instalação experimental para pesquisas em Internet do Futuro no Brasil, sendo federado com *testbeds* europeus em ambos os níveis, tanto no de conectividade física quanto no arcabouço de controle e monitoração.

O projeto FIBRE terminou em Outubro de 2014, após uma duração de 30 meses, mais um período de extensão. Em 2015 as instituições brasileiras participantes assumiram a infraestrutura legada do FIBRE para começar a oferecer o *testbed* como um serviço. Um novo modelo de governança, regras de admissão, políticas e processos de operação de serviços foram propostos. No entanto, a marca do *testbed* e o nome permaneceram os mesmos, mas com um novo significado: *Future Internet Brazilian Environment for Experimentation*.

Capítulo 3

Os centros de dados e suas topologias de rede

Um centro de dados é uma estrutura física que abriga servidores, uma estrutura de rede para interconectá-los e a infraestrutura necessária para seu funcionamento (energia elétrica, refrigeração, segurança), além das aplicações necessárias para os serviços. Atualmente, os centros de dados fornecem o *backbone* computacional para a Internet [29].

As redes dos centros de dados apresentam topologias variadas, em geral, derivadas da topologia em árvore. De acordo o estudo apresentado em [9], a utilização dos enlaces nas camadas de borda (*edge*) e agregação da rede é relativamente baixa e exibe pouca variação, enquanto a utilização dos enlaces no núcleo (*core*) é alta e com variação significativa ao longo de um dia. Alguns enlaces no núcleo apresentam congestionamento[1] persistente, apesar de existir capacidade disponível suficiente para aliviar tal congestionamento. Ainda, a redundância da rede é apenas 40% efetiva na redução do impacto mediano causado por uma falha [28].

A topologia da rede e a forma como o tráfego é distribuído ao longo de seus enlaces constituem fatores chave para o desempenho nas comunicações dos centros de dados.

Topologia é o arranjo pelo qual estão conectados os nós de uma rede, seja considerando os enlaces físicos ou lógicos. Desde os primórdios das comunicações de rede, as topologias desempenham um papel importante na concretização de seu objetivo, moldando através de suas características alguns comportamentos adotados pelos protocolos de comunicação de forma a atender os requisitos de projeto. Duas topologias notáveis são a topologia em

[1] A expressão *enlace congestionado* é utilizada convencionalmente para enfatizar que o congestionamento afeta a comunicação entre os elementos localizados nos extremos do enlace. Deve estar claro que o congestionamento ocorre de fato nas filas (*buffers*) das portas dos elementos de rede (comutadores / roteadores).

estrela e a topologia em malha.

Os centros de dados tem evoluído rapidamente nos últimos anos, buscando constantemente atender a demanda, cada vez maior e variada [65]. Eles concentram a infraestrutura para execução de aplicações heterogêneas, visando flexibilidade, eficiência e redução de custos. Assim, seu desenvolvimento tem fomentado o surgimento de topologias mais específicas, com foco principalmente na redução de custos de implantação e operação. As topologias hierárquicas, como as baseadas em árvore multi raízes são as mais populares nos projetos dos centros de dados [30, 63]. Essas topologias serão explicadas nas próximas Seções.

As topologias estão intimamente relacionadas às técnicas utilizadas para o transporte das comunicações, especialmente o roteamento e o encaminhamento de pacotes, como será visto no próximo capítulo. Os benefícios de inovação e flexibilidade que as RDS oferecem são grandes incentivadores para sua implantação nos centros de dados.

3.1 Topologias básicas

Topologias básicas são relativamente simples de serem construídas, enquanto que topologias mais complexas podem ser criadas pela combinação de topologias básicas. As topologias básicas de rede exibem características que também podem ser encontradas em segmentos das topologias mais complexas. A seguir, descrevemos duas topologias básicas presentes em quase todas as topologias complexas.

3.1.1 Estrela

A topologia em estrela é caracterizada por um nó central utilizado para interconectar todos os nós periféricos. Como exemplo, temos um comutador interconectando os *hosts* em uma rede. Se esse comutador for não bloqueante e a capacidade de suas interfaces maior ou igual à capacidade das interfaces de rede dos *hosts*, a comunicação entre qualquer par de *hosts* poderá ser realizada à taxa máxima alcançável pelas suas respectivas interfaces, como se os *hosts* estivessem interconectados ponto a ponto.

Simplicidade e facilidade de projeto, instalação e manutenção são as principais vantagens das topologias em estrela. Não obstante, as topologias em estrela também apresentam algumas desvantagens importantes. O tamanho da rede será limitado pelo número de portas do nó central. Caso o nó central apresente falha, toda a rede entrará em co-

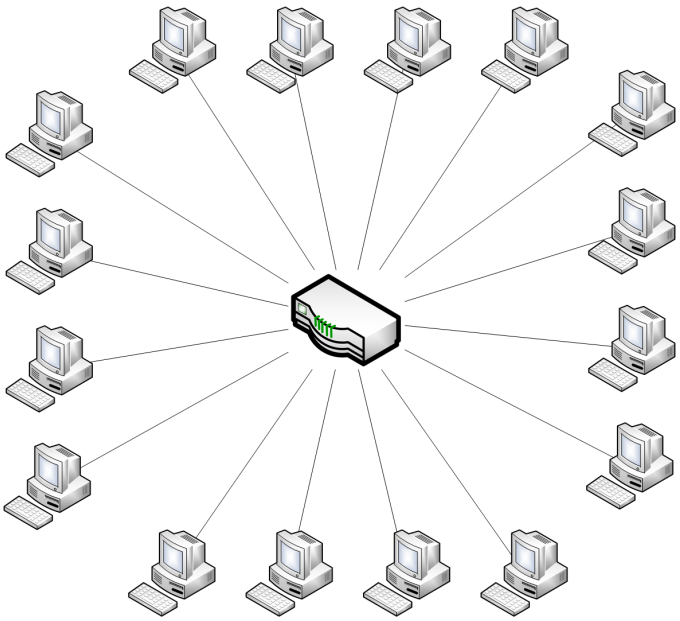

Figura 3.1: Exemplo de uma topologia em estrela

lapso, resultando na falta de conectividade entre seus *hosts* ou severa degradação nas comunicações, dependendo da falha apresentada.

3.1.2 Malha

Uma topologia em malha apresenta ao menos dois nós com dois ou mais caminhos entre eles, fornecendo assim, caminhos redundantes para serem utilizados no caso de falha em algum dos outros caminhos. Essa característica descentralizada visa superar uma das principais desvantagens encontradas nas redes centralizadas (como estrela e árvore) onde a falha no equipamento central ocasiona a perda de conectividade de muitos nós ou até mesmo da rede como um todo.

O projeto e a implantação de uma rede em malha é um pouco mais difícil, devido à ampla possibilidade de grafos arbitrários. A escolha do grau de conectividade envolve a ponderação de custos e complexidade.

Uma topologia em malha onde há a ligação direta entre todos os pares de nós recebe o nome de *rede totalmente conectada*, *topologia completa* ou ainda *topologia de malha completa*. As topologias de malha completa proporcionam um altíssimo grau de confiabilidade, por causa da conexão direta entre cada par de nós e pelos múltiplos caminhos existentes entre cada par de nós. Essa grande densidade de interconexões também acaba aumentando significativamente o custo total da rede.

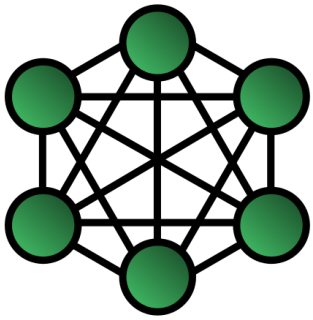

Figura 3.2: Exemplo de uma topologia de malha completa

A partir de uma topologia de malha completa, pode-se derivar qualquer outra topologia de rede através da desconexão de forma conveniente dos enlaces entre alguns nós.

3.2 Topologias em centros de dados

A arquitetura da rede que interconecta os servidores em um centro de dados é baseada, tipicamente, em uma topologia hierarquizada em forma de árvore [5].

A hierarquia de conexão lembra uma árvore de cabeça para baixo, com os comutadores interconectados em diferente níveis, da borda da rede (folhas) até o núcleo (raiz). Servidores em um *rack* são conectados a um comutador de borda que, geralmente, situa-se no topo do *rack* e também é referido como *top-of-rack switch*. Comutadores de vários *racks* são conectados a comutadores de agregação que, por sua vez, são conectados a um comutador de núcleo (*core switch*).

Para garantir resiliência, pode existir mais de uma conexão entre dois elementos da rede e, consequentemente, múltiplos caminhos possíveis para o tráfego de dados entre um par de servidores.

Mesmo assim, na topologia em árvore com sobrescrição, quanto maior o nível dos elementos de rede, isto é, quanto mais próximos do núcleo da rede, maior o índice de sobrescrição dos enlaces e, consequentemente, a probabilidade de congestionamento nos enlaces.

O índice de sobrescrição de uma topologia de comunicação pode ser definido como a razão entre taxa de transferência agregada alcançável pelos nós terminais no pior caso e a taxa de transferência total da bissecção dessa topologia.

Um índice de sobrescrição de 1:1 indica que todos os nós podem, eventualmente,

se comunicar com qualquer outro nó da rede utilizando a taxa de transferência total de suas interfaces de rede, ou seja, este é o melhor caso onde estão presentes na rede todos os equipamentos e enlaces necessários para que os nós exerçam todo o seu poder de comunicação.

Um valor de sobrescrição de 5:1 indica que apenas 20% de toda a taxa de transferência de um nó estará disponível para alguns padrões de comunicação. Ou seja, o número total de equipamentos e enlaces de rede é limitado, de forma que apenas uma fração da capacidade total da interface de rede de um nó será efetivamente utilizada.

Arquiteturas de redes de centros de dados geralmente empregam valores de sobrescrição da ordem de 2,5:1 a até 8:1 [5], de forma a reduzir o custo total de sua implantação.

As topologias a seguir são extensões da topologia em árvore e, como será visto no próximo capítulo, possibilitam a utilização de técnicas de roteamento por múltiplos caminhos para alcançar um melhor uso da capacidade disponível na rede em comparação ao uso máximo alcançável por topologias (lógicas) de caminho único.

3.2.1 Topologia em árvore com múltiplas raízes comum

A topologia em árvore com múltiplas raízes comum é um padrão bastante adotado, estando presente na maioria dos centros de dados tradicionais, e está representado na Figura 3.3. Nessa topologia, múltiplos comutadores de núcleo são utilizados, de forma a oferecer largura de banda total entre qualquer par de servidores, ou seja, fisicamente não existe sobrescrição. Geralmente, a conexão entre os comutadores está estruturada em três níveis: borda (*edge*), agregação (*aggregation*) e núcleo (*core*).

3.2.2 Topologia em árvore com múltiplas raízes do tipo folha-espinha (*leaf-spine*)

A topologia folha-espinha (*leaf-spine*), mostrada na a Figura 3.4, é utilizada em centros de dados mais modernos, onde a maior demanda é por desempenho da rede de comunicação [15], pois fornece taxa de transferência altamente escalável, de forma uniforme e previsível através de múltiplos caminhos de mesma distância, isto é, caminhos que possuem a mesma quantidade de nós intermediários [7]. Na topologia folha-espinha, a conexão entre os comutadores está estruturada em apenas dois níveis: folha (*leaf*) e espinha (*spine*).

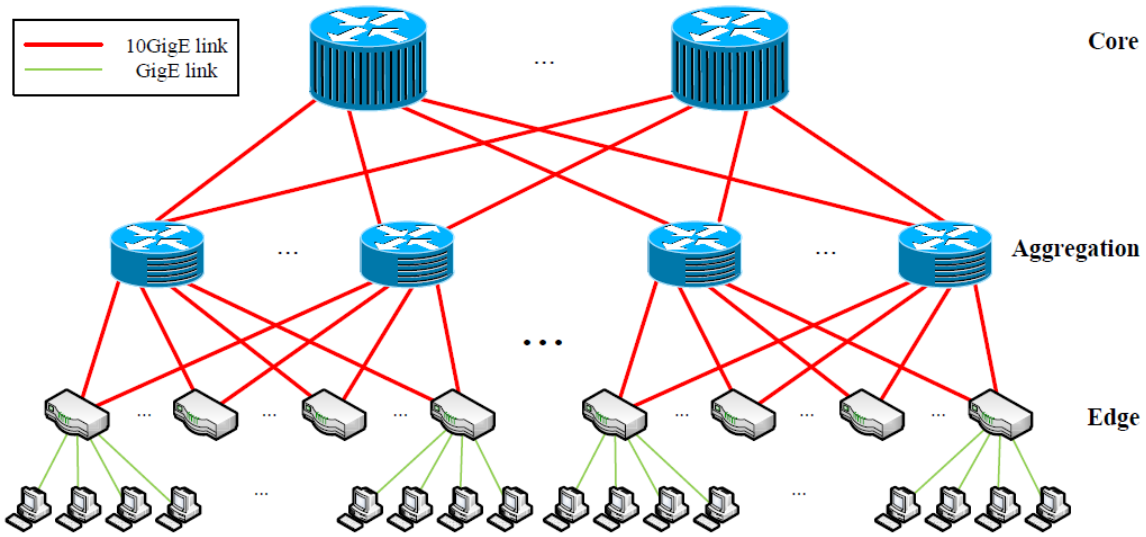

Figura 3.3: Exemplo de uma topologia hierárquica em árvore com múltiplas raízes do tipo comum [6]

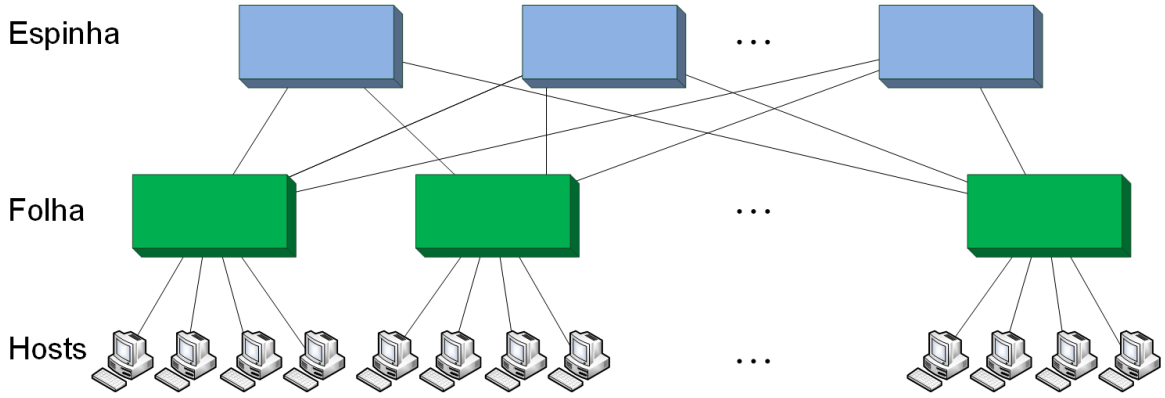

Figura 3.4: Exemplo de uma topologia hierárquica em árvore com múltiplas raízes do tipo folha-espinha (*leaf-spine*)

3.2.3 Topologia em árvore com múltiplas raízes do tipo *fat-tree*

Uma topologia *fat-tree* [5] é mais um dos arranjos possíveis de uma topologia em árvore com múltiplas raízes. Mais especificamente, a *fat-tree* é uma instância especial de uma topologia Clos [55]. A Figura 3.5 ilustra um exemplo de *fat-tree* com $k = 4$.

Em uma *fat-tree*, existem k *pods*, contendo cada um duas camadas de $\frac{k}{2}$ comutadores. Cada comutador na camada inferior (camada de borda) possui k portas e está diretamente conectado a $\frac{k}{2}$ servidores. Cada uma das $\frac{k}{2}$ portas restantes está ligada a $\frac{k}{2}$ das k portas presentes na camada superior (camada de agregação) da hierarquia. Existem $(\frac{k}{2})^2$ comutadores de núcleo com k portas cada. Cada comutador de núcleo tem cada uma de suas portas conectadas a um dos k *pods*. Desta forma, a *iésima* porta de qualquer comutador

de núcleo está ligada ao *pod* i de forma que as portas consecutivas de cada comutador na camada de agregação de cada *pod* estão conectadas ao núcleo através de $(\frac{k}{2})$ passos.

Ao considerar uma topologia *fat-tree* constituída por comutadores com k portas, organizados em três níveis, borda (*edge*), agregação (*aggregation*) e núcleo (*core*), a quantidade de comutadores será igual a $\frac{5}{4}k^2$ com um máximo de até $\frac{k^3}{4}$ servidores [5]. As *fat-trees* fornecem um alto grau de diversidade de caminhos e são não bloqueantes por rearranjo[2].

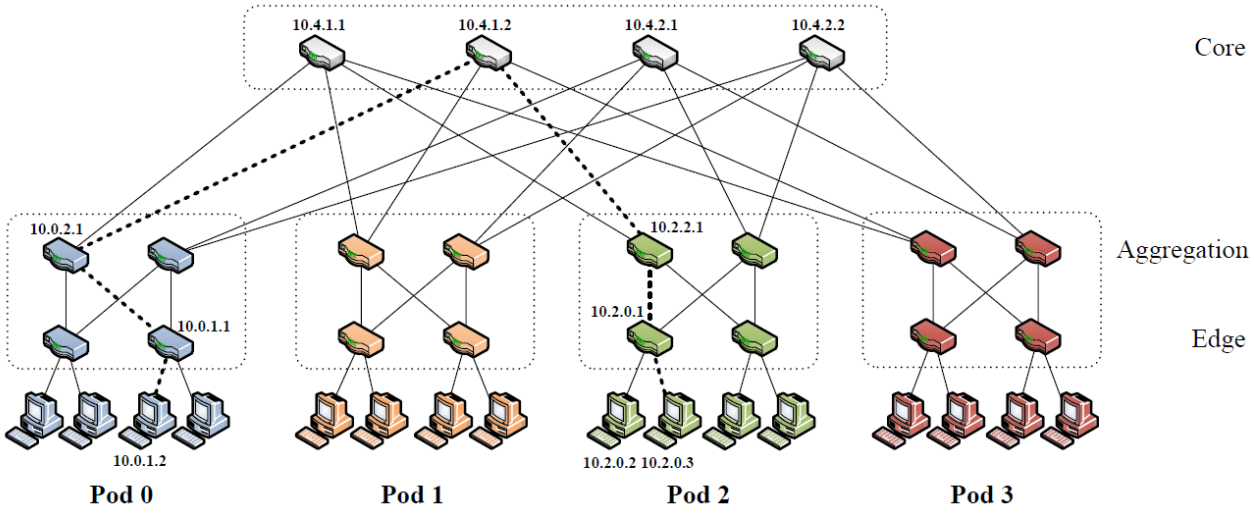

Figura 3.5: Exemplo de uma topologia hierárquica em árvore com múltiplas raízes do tipo *fat-tree* com $k = 4$ [5]

3.2.4 Topologia aleatória

Uma topologia criada de forma aleatória pode apresentar propriedades interessantes. A Jellyfish [78] é uma topologia não estruturada, concebida para interconectar a rede aleatoriamente, e permitir a expansão dos centros de dados de forma incremental e sem barreiras. A Jellyfish chega a suportar 25% mais servidores que uma topologia *fat-tree* quando composta por alguns milhares de nós.

A Figura 3.6 mostra uma topologia aleatória Jellyfish, com o número de elementos igual ao apresentado na topologia *fat-tree* da Figura 3.5. Cada anel "concêntrico" contém servidores alcançáveis a partir de qualquer outro servidor arbitrário dentro do número de saltos que é indicado na figura.

[2]Não bloqueantes por rearranjo significa que, considerando as possíveis permutações entre pares de servidores origem-destino, sempre será possível conectar uma origem a um destino, porém, com a necessidade de reorganizar os caminhos já utilizados por um ou mais pares de servidores origem-destino.

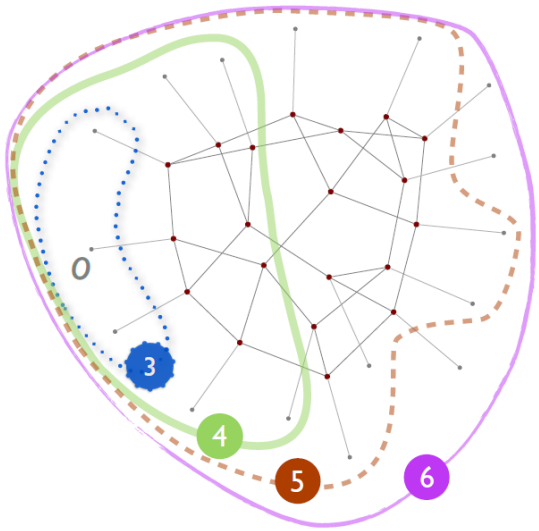

Figura 3.6: Grafo de uma topologia aleatória Jellyfish, contendo 16 servidores (extremidades) e 20 comutadores de 4 portas [78]

Capítulo 4

Técnicas para escolha do melhor caminho

Um comutador Ethernet tem como missão principal encaminhar os pacotes ingressantes em cada uma de suas entradas para a saída correta, onde eles alcançarão seu destino. Para cada enlace ao qual está conectado, um comutador executa o protocolo de enlace adequado para comunicar-se com o nó da outra extremidade. O comutador consulta sua tabela de encaminhamento para decidir como um pacote será encaminhado.

Com as redes interconectadas, é possível que existam várias maneiras de se chegar de um ponto a outro. Encontrar um caminho ou rota adequados em uma rede é um dos problemas fundamentais do uso de redes. Tais caminhos devem ser eficientes, livres de ciclos (*loops*) e capazes de responder à dinâmica das redes (inclusão e exclusão de nós e enlaces, falhas, etc) [72].

4.1 Técnicas de roteamento simples

Nas topologias de rede que apresentam mais do que um caminho possível entre dois nós torna-se necessária a construção de um caminho único (lógico) para realizar o encaminhamento entre cada par de nós origem-destino, pois a presença do ciclo (*loop*) induzido pelos múltiplos caminhos implica no surgimento de tempestades de difusão (*broadcast storms*) que deixam o segmento de rede inoperante. Uma tempestade de difusão ocorre quando, por exemplo, inicialmente um único quadro de difusão é enviado para todos os nós vizinhos, e retorna (depois de ser difundido pelo nó vizinho) pelo caminho alternativo e, então, o ciclo se inicia novamente e se repete de forma perpétua, pois não existe um mecanismo de proteção como o TTL, por exemplo, no cabeçalho de um quadro.

Para evitar esse problema, um algoritmo de árvore geradora (*spanning tree*) é utili-

zado para derivar uma topologia sem ciclos a partir da topologia atual da rede. Mais especificamente, a Árvore Geradora Mínima (*Minimum Spanning Tree* (MST)) [88] é calculada. O STP [41] realiza o bloqueio das portas que não fazem parte da árvore calculada, deixando um único caminho disponível entre dois nós ao longo desta árvore.

4.2 Técnicas de roteamento definido pela fonte

No roteamento definido pela fonte, o *host* de origem (fonte) fornece toda a informação de topologia (caminho) necessária para comutar um pacote ao logo da rede. Essa informação é colocada no cabeçalho do pacote pelo *host* de origem e, ao recebê-lo, cada elemento da rede responsável pelo encaminhamento inspeciona o campo adequado no cabeçalho para decidir para qual porta de saída o pacote será encaminhado.

Então, o *host* de origem necessita conhecer toda a informação a respeito da topologia da rede para formar um cabeçalho que contenha todo o direcionamento correto para cada comutador no caminho. Uma desvantagem dessa abordagem é o tamanho extremamente variável do cabeçalho [72].

4.3 Técnicas de roteamento multi caminho

Nessa Seção são apresentadas algumas técnicas desenvolvidas para aumentar a eficiência no uso das topologias onde múltiplos caminhos entre dois nós estão presentes.

O alcance da maior taxa de transferência possível em uma rede com múltiplos caminhos requer a distribuição homogênea da carga entre todos os caminhos disponíveis. Topologias que oferecem múltiplos caminhos para a comunicação entre dois nós, como as topologias em árvore multi raízes por exemplo, podem ter seu potencial desperdiçado com a escolha de um algoritmo que suporte apenas um único caminho.

Algumas técnicas específicas para uso em centros de dados foram desenvolvidas de forma a aproveitar melhor as características diferenciadas presentes nessas instalações. A rede dos centros de dados pode acabar sendo o gargalo das comunicações, caso boas técnicas de roteamento e encaminhamento dos fluxos não sejam adotadas [30].

O ECMP [83, 40] é, sem dúvida, a técnica mais utilizada. Também será comentado aqui o MPTCP [44] , que foi proposto para permitir que *hosts* utilizem simultaneamente conexões por mais de uma de suas múltiplas interfaces de rede (*multi homed*). Estudos

recentes sobre a aplicação do MPTCP indicam sua contribuição na melhora da eficiência de encaminhamento dos centros de dados [74].

4.3.1 Equal-Cost Multi-Path - ECMP

Mecanismos de encaminhamento baseados em Múltiplos Caminhos de Mesmo Custo (*Equal-Cost Multi-Path* - ECMP) são comumente empregados para alcançar uma melhor utilização dos enlaces e a consequente melhoria na taxa de transferência da rede.

O ECMP realiza o balanceamento dos fluxos de forma estática. A divisão da carga é alcançada pela escolha de qual porta será utilizada para encaminhar cada fluxo de acordo com um *hash* calculado a partir de determinados campos do cabeçalho dos pacotes ingressantes. O *hash* de alguns campos específicos do cabeçalho de um pacote garante que todos os pacotes pertencentes a um mesmo fluxo irão seguir pelo mesmo caminho na rede, evitando assim o reordenamento desnecessário de pacotes. Esta escolha não leva em conta a taxa de transferência de cada fluxo, o que pode levar ao congestionamento da rede até mesmo para sequências de comunicação simples, isto é, entre poucos *hosts* [6].

Comutadores implementam o ECMP com base na criação de grupos multi caminho que representam uma lista de portas de saída de "mesmo custo" para um prefixo de destino (fluxo). Cada porta de saída corresponde a um ou mais dos múltiplos caminhos disponíveis para se alcançar o destinatário. A quantidade máxima de múltiplos caminhos considerados pelo comutador nas implementações atuais é de 128 para os comutadores de última geração [8] e apenas 16 para comutadores com especificações mais modestas [16].

No entanto, por depender de *hash* estático dos fluxos e escolha aleatória dos enlaces de destino baseada nos *hashes*, redes que implementam tais mecanismos podem sofrer perdas de taxa de transferência substanciais devido, principalmente, à colisões de fluxos onde dois ou mais fluxos dividem a capacidade de um mesmo enlace enquanto outros enlaces permanecem ociosos.

Ainda, a escolha aleatória da porta de saída para cada fluxo realizada individualmente pelos comutadores pode ocasionar seu encaminhamento por trechos mais congestionados do caminho, reduzindo a taxa máxima alcançada em relação à que poderia ser efetivamente alcançada caso esses mesmos fluxos fossem encaminhados através dos caminhos menos congestionados. Outra limitação do ECMP é que ele depende de caminhos "iguais", com mesmo custo e mesma taxa de transferência para desempenhar bem seu papel.

4.3.2 TCP Multi Caminho - *Multipath* TCP

O *Multipath* TCP (MPTCP) [44] é uma solução a nível da camada de transporte que propõe a criação de vários sub fluxos TCP para cada fluxo TCP original, indo de encontro à ideia tradicional nas redes de centros de dados que dita "um fluxo, um caminho". Cada sub fluxo entre um *host* emissor e outro receptor pode ser criado utilizando o mesmo par de endereços IP (origem e destino) e portas TCP diferentes, ou simplesmente endereços de IP distintos caso pelo menos um dos *hosts* possua mais de uma interface, cada uma com seu próprio endereço de rede (IP).

O MPTCP proporciona, em uma curta escala de tempo, o balanceamento de carga distribuído, quando combinado com o uso do ECMP. O que, por sua vez, alcança uma melhor utilização dos caminhos redundantes disponíveis em uma topologia de rede de centro de dados, oferecendo um melhor desempenho sem custos adicionais.

Em [74] é apresentada uma proposta e prova de conceito para a aplicação do MPTCP em redes de centros de dados. A proposta apresentada depende do ECMP para realizar o *hash* dos sub fluxos e espalhá-los pelos múltiplos caminhos, de forma a proporcionar um melhor balanceamento da rede. Através da subdivisão de cada fluxo, aumenta-se sua segmentação melhorando a probabilidade de distribuição de pacotes por caminhos diferentes e reduzindo as colisões de fluxos duradouros, melhorando a eficiência total da rede.

Para um novo fluxo, pode parecer convidativo aumentar indefinidamente o número de sub fluxos para alcançar um espalhamento mais equilibrado pelos múltiplos caminhos da rede, porém, cada novo sub fluxo consome recursos como CPU e memória nos *hosts* e aumenta o *overhead* nos pacotes entregues às redes, reduzindo o ganho de eficiência efetivamente alcançado.

Uma desvantagem apresentada pelo MPTCP em relação a outras técnicas de roteamento multi caminho é a necessidade de alteração no sistema operacional dos *hosts* para suportar as novas funcionalidades implantadas entre as camadas de rede e de transporte.

4.4 Algoritmos distribuídos vs. centralizados

Algoritmos distribuídos de roteamento são amplamente empregados, de forma a proporcionar autonomia aos elementos da rede e aumentar a resiliência da rede. No entanto, os protocolos distribuídos podem levar um tempo demasiadamente longo para convergir

a um estado adequado, principalmente durante a ocorrência de falhas, e, assim, medidas "externas" tem sido adotadas para melhorar o desempenho da rede através da engenharia de tráfego.

Algoritmos de encaminhamento centralizados começaram a surgir como uma forma de auxiliar os algoritmos distribuídos otimizando sua funcionalidade. Outras propostas de algoritmos centralizados surgiram até o advento das redes definidas por *software*, onde ultimamente o controlador da rede é o cérebro que orquestra os elementos encaminhadores, de forma intrinsecamente centralizada.

Propostas para superar as limitações dos protocolos atuais tem surgido, empregando técnicas variadas que vão desde a realocação dinâmica de fluxos com base em seu tamanho [6] até o espalhamento aleatório de pacotes [21, 15].

Capítulo 5

Equalize: encaminhamento equilibrado

Nesse trabalho, é proposto um mecanismo de encaminhamento simples, porém efetivo, baseado em RDS para a distribuição dos fluxos de dados pela rede através dos diversos caminhos disponíveis entre dois *hosts*, de forma a alcançar a plena utilização dos recursos oferecidos pela rede. Esse arcabouço foi batizado de "Equalize" pela sua natureza equalizadora na distribuição do tráfego ao longo da rede. O Equalize realiza o balanceamento automático do tráfego pelos enlaces disponíveis, de forma transparente para os *hosts* da rede, o que implica em:

1. Melhor aproveitamento dos enlaces disponíveis, como se toda a rede fosse composta por uma única matriz de comutação.

2. Resiliência, pois, em caso de falha em algum enlace, a conectividade será mantida através dos demais enlaces existentes.

Embora o Equalize possa ser aplicado a qualquer topologia gerenciável de forma centralizada, nessa dissertação, sua aplicação está focada nas topologias e métodos de roteamento e encaminhamento utilizados nos centros de dados.

O Equalize é composto por uma suíte de aplicações de rede, sobre a plataforma fornecida pelo controlador de rede Beacon [25], cada uma com as atribuições descritas a seguir e detalhadas na Seção 5.5:

- A aplicação de Descoberta da Topologia garante a descoberta de enlaces entre comutadores adjacentes. Ela também é responsável pelo relato de falhas de conexão nos enlaces, mantendo uma base de adjacências atualizada para refletir o estado real da topologia da rede;

- A aplicação Árvore Geradora (*Spanning Tree*) mantém a rede livre de ciclos para os pacotes de difusão (*broadcast*) e *multicast*;
- A aplicação de Coleta de Medições periodicamente solicita, reúne e processa as medições das portas de cada comutador;
- A aplicação de Roteamento calcula os caminhos mais curtos (menos utilizados) entre todos os comutadores através do algoritmo de Dijkstra adaptado;
- A aplicação de Aprendizado-Consulta-e-Encaminhamento trata os pacotes que ainda não possuem alguma ação associada em uma entrada na tabela dos comutadores (novo fluxo), enviando instruções para adicionar a ação adequada a tal fluxo na tabela dos comutadores pertinentes.

A figura 5.1 mostra a arquitetura do Equalize.

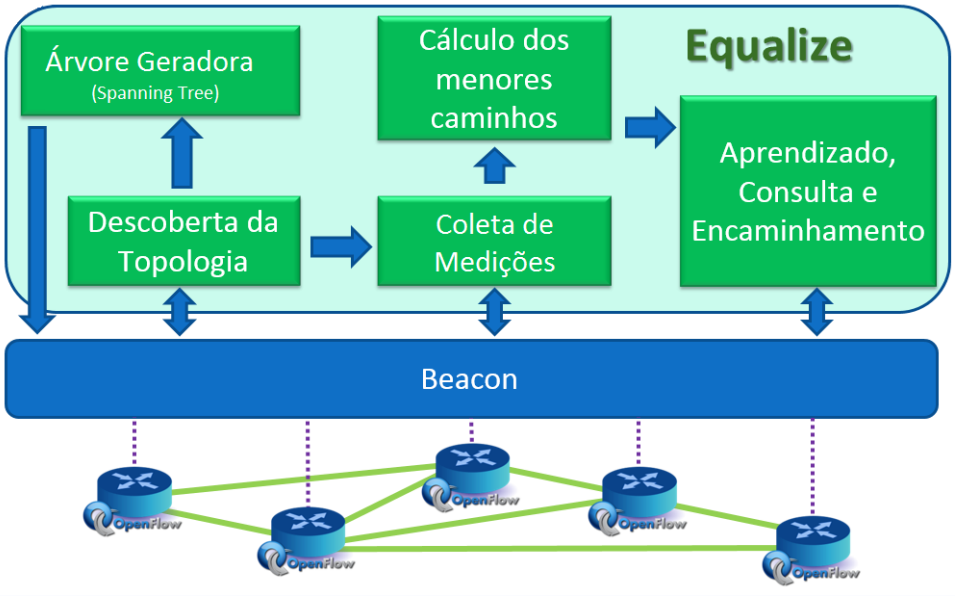

Figura 5.1: Arquitetura do Equalize

5.1 Lógica de processamento

O encaminhamento dos pacotes é baseado no endereço de camada 2 (MAC), proporcionando um domínio comum para toda a rede e permitindo o posicionamento de *hosts* em qualquer ponto da rede, e posterior reposicionamento (migração) caso necessário, dependendo apenas de procedimento padrão de expiração do *cache* ARP ou do envio de um ARP gratuito para a atualização da sua posição na rede.

O controlador "descobre" e mantém um registro atualizado da topologia da rede através dos métodos incluídos por padrão no controlador Beacon. Essa descoberta é feita por meio de pacotes LLDP criados especificamente para a descoberta de elementos na camada de enlace.

Na sequência, o algoritmo calcula a árvore geradora e comanda o bloqueio (marcação do bit **no_flood** no mapa de bits) das portas que não fazem parte da árvore geradora, considerando apenas portas de enlaces entre comutadores[1]. Ele também calcula os menores caminhos entre os comutadores, realiza o aprendizado de quais *hosts* estão conectados a quais comutadores identificando quais endereços MAC aparecem nas portas dos comutadores que não estão conectadas a outros comutadores. Ao perceber a presença de um novo fluxo, o controlador "aprende" que o novo MAC de origem está conectado ao comutador de origem caso esse MAC ainda não seja conhecido, em seguida, pesquisa o comutador conectado ao *host* de destino, e atualiza as tabelas dos comutadores pertencentes ao menor caminho para realizarem o encaminhamento dos pacotes desse novo fluxo.

O algoritmo de balanceamento proposto realiza o cálculo do menor caminho entre dois comutadores considerando como custo (ou utilização l) a fração de utilização de cada enlace:

$$l = \frac{B_u}{C_{max}}, \tag{5.1}$$

onde B_u é a taxa de transferência efetivamente utilizada (Banda utilizada) e C_{max} é a Capacidade máxima do enlace.

O resultado dos menores caminhos entre todos os comutadores da rede é obtido através do cálculo do menor caminho entre um par de comutadores utilizando o algoritmo de Dijkstra adaptado (ver Seção 5.4), realizado repetidamente para cada par de comutadores.

Ao considerar o equilíbrio no uso dos enlaces como um dos objetivos do encaminhamento de fluxos, consegue-se melhorar o uso geral da rede. Tomando a decisão por onde encaminhar os fluxos de forma determinística e centralizada, o Equalize também proporciona facilidade e agilidade no gerenciamento da rede e na identificação e tratamento de falhas.

É importante ressaltar que a proposta não requer mudança em algoritmos ou protocolos nos *hosts* nem nos elementos de rede. Utiliza-se a tecnologia existente de comutadores

[1]Em comutadores OpenFlow, a marcação do bit **no_flood** no mapa de bits para bloqueio de pacotes de difusão (*broadcast*) não implica no bloqueio da porta para transmissão de pacotes *unicast* ou *multicast*. Essa é uma aplicação mais flexível e diferente do comportamento apresentado pelo STP [41] nos comutadores convencionais que realiza o bloqueio de todo o tráfego na porta.

programáveis através do protocolo OpenFlow [2] e um controlador de rede centralizado para alcançar o objetivo de auto balanceamento da rede, resultando na melhora geral da eficiência no encaminhamento de fluxos.

Nas seções a seguir, o conceito de auto-balanceamento será explicado, bem como as técnicas utilizadas para sua aplicação na melhoria do uso dos enlaces.

5.2 O conceito de auto-balanceamento

A distribuição pacote a pacote pelos caminhos disponíveis entre dois *hosts* é a forma mais eficiente de garantir o equilíbrio do uso desses caminhos. Porém, caminhos com atrasos diferentes resultam na chegada dos pacotes fora de ordem ao *host* de destino. Esse reordenamento dos pacotes causa problemas para algoritmos que dependem da entrega ordenada dos mesmos, como é o caso por exemplo do TCP, ocasionando degradação no uso dos enlaces com a redução da vazão útil (*goodput*).

Para evitar esse efeito indesejado, sem modificar os protocolos tradicionalmente em uso, a alternativa para encaminhamento dos pacotes que segue com maior granularidade é o uso do agrupamento desses pacotes em fluxos. Desta forma, consideramos como um mesmo fluxo pacotes que possuem o mesmo *host* e porta TCP/UDP de origem e também mesmo *host* e porta TCP/UDP de destino.

Ao distribuir tais fluxos pelos múltiplos caminhos existentes entre dois *hosts*, consegue-se manter o encaminhamento da rede sempre direcionado ao equilíbrio no uso dos enlaces como um todo, com a penalidade de incorrer no aumento do número de fluxos presentes na tabela de fluxos.

5.3 Utilização em um enlace

Através da monitoração do tráfego que é transmitido em cada porta de um comutador direcionado para outro comutador, torna-se possível identificar a condição de uso da porta. Ao realizar a divisão desse tráfego pela capacidade máxima da porta (definida manualmente ou automaticamente durante o estabelecimento do enlace) a razão de utilização do enlace é obtida, a qual reflete diretamente a carga do enlace.

Com a informação da topologia da rede e da utilização nas portas que interconectam os elementos responsáveis pelo encaminhamento dos pacotes, é construído um **grafo di-**

recionado [86] com os elementos sendo os vértices e suas interconexões as arestas. Cada aresta tem como propriedade a razão de utilização da porta de origem do enlace.

Esse grafo será a matéria prima para o cálculo dos menores caminhos entre os elementos, utilizando como métrica de custo a razão de utilização.

Note que a técnica proposta não retorna como resultados todos os menores caminhos possíveis entre dois *hosts* considerando a quantidade de saltos entre eles como métrica. Apenas um único caminho entre dois *hosts* é calculado. Ou seja, dentre todos os caminhos possíveis, apenas aquele que possuir a menor utilização fim a fim será escolhido e, caso exista mais de um caminho com o mesmo valor mínimo de utilização, um deles será escolhido aleatoriamente.

A implementação realizada utiliza o protocolo OpenFlow versão 1.0 [2] para comunicação entre o controlador e os comutadores. Infelizmente, essa versão não oferece mecanismos para informar ao controlador a taxa de transferência efetivamente utilizada B_u por cada porta a cada instante. Desta forma, essa utilização precisa ser inferida pelo controlador utilizando-se a diferença ΔTX entre a quantidade atual de *bytes* transmitidos TX_{atual} pela porta e a quantidade coletada na medição anterior TX_{hist}, dividida pelo intervalo de tempo Δt entre as medições:

$$B_u = \frac{\Delta TX}{\Delta t}, \tag{5.2}$$

onde:

$$\Delta t = t_{atual} - t_{hist} \tag{5.3}$$

Tal inferência realizada pelo controlador implica na adição de erro, devido principalmente, a realização da marcação do intervalo de tempo Δt entre as medições no momento em que a medição chega ao controlador e não no momento em que ela é realizada no comutador. Para reduzir a influência do erro nessa estimativa realizada no lado do controlador, calcula-se a taxa de transferência utilizada média B'_u, composta pela taxa de transferência utilizada atual $B_{u_{atual}}$ e pela taxa de transferência utilizada coletada na medição anterior$B_{u_{hist}}$.

$$B'_u = \frac{B_{u_{atual}} + B_{u_{hist}}}{2} \tag{5.4}$$

5.4 O algoritmo de Dijkstra adaptado

Conforme explicado na Seção anterior, um **grafo direcionado** com a representação dos elementos responsáveis pela comutação na rede, suas interconexões e a utilização em cada interconexão serão utilizados para encontrar o menor caminho entre dois *hosts*. O caminho encontrado será, portanto, o que possui a menor razão de utilização fim a fim.

Encontrar o menor caminho entre dois *hosts* trata-se, em teoria de grafos, do problema clássico de encontrar o caminho mínimo entre todos os pares de vértices (*All Pairs Shortest Path* (APSP)).

O algoritmo de Dijkstra original realiza a iteração sobre os vértices do grafo, ordenados pela menor distância conhecida até o momento para o caminho formado pelas arestas do vértice de origem ao vértice atual, atualizando a distância desse caminho para o menor valor resultante da soma da distância de qualquer vizinho com o custo da aresta que o conecta a esse vizinho [20].

Ou seja, dado um grafo direcionado $G = (V, E)$ formado por um conjunto V de vértices interconectados por um conjunto E de arestas onde cada aresta está associada a um custo w, o custo total do menor caminho $w(p)$ entre dois vértices (u, v) pode ser definido como:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i), \tag{5.5}$$

onde o custo $w(p)$ do caminho $p = \langle v_0, v_1, ..., v_k \rangle$ é a soma dos custos das arestas presentes nesse caminho. Essa soma pode ser minimizada pela busca do custo do menor caminho $\delta(u, v)$ considerando:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \rightsquigarrow^p v\} & \text{se existe um caminho de } u \text{ até } v, \\ \infty & \text{caso contrário.} \end{cases} \tag{5.6}$$

No Equalize, ao utilizar uma razão como métrica de custo, o algoritmo de Dijkstra precisa ser adaptado para considerar como distancia total do caminho (custo do caminho) o maior valor de utilização presente nesse caminho, ao invés da soma dos custos ao longo do caminho.

Este é um problema de Min-Max, que consiste em encontrar um caminho cujo maior peso das arestas seja mínimo [19]. Além do algoritmo de Dijkstra ser empregado nos

protocolos de roteamento da Internet, ele também é fácil de ser adaptado para computar o melhor caminho utilizando uma função Min-Max ao invés da original Min-Sum. E por esses motivos ele foi escolhido para ser utilizado nesse trabalho.

Desta forma, o cálculo do caminho com menor utilização entre qualquer par de *hosts* da rede é realizado utilizando-se o algoritmo de Dijkstra **adaptado** mostrado no Algoritmo 1.

Por considerar como métrica de custo uma razão (a razão de utilização) ao invés de um valor inteiro (como o número de saltos), o cálculo do caminho menos utilizado entre dois vértices deve ser ajustado para considerar a maior utilização ao longo do caminho como sendo a utilização total desse caminho, ao invés de considerar a soma das utilizações.

Ou seja, dado um grafo direcionado $G = (V, E)$ formado por um conjunto V de vértices interconectados por um conjunto E de arestas onde cada aresta está associada a uma utilização l, a utilização total do menor caminho $l(p)$ entre dois vértices (u, v) pode ser definida como:

$$l(p) = \max\{l(v_{i-1}, v_i)\}, \tag{5.7}$$

onde a utilização total $l(p)$ do caminho $p = \langle v_0, v_1, ..., v_k \rangle$ é a maior utilização encontrada entre as arestas presentes nesse caminho. O Algoritmo de Dijkstra adaptado encontra o melhor caminho pela busca daquele que possui a menor utilização $\delta(u, v)$ considerando:

$$\delta(u, v) = \begin{cases} \min\{l(p) : u \rightsquigarrow^p v\} & \text{se existe um caminho de } u \text{ até } v, \\ \infty & \text{caso contrário.} \end{cases} \tag{5.8}$$

Esse ajuste é a modificação realizada no algoritmo de Dijkstra original, resultando no algoritmo de Dijkstra **adaptado** utilizado nesse trabalho.

No Algoritmo 1, temos que:

- G é o grafo direcionado contendo todos os vértices da rede, suas adjacências e respectivos custos;
- s é o nó de origem;
- d é o nó de destino;
- Q é o conjunto contendo tuplas com: o custo l do caminho, o nó v a ser testado

como nó final do caminho e a lista P com nós pertencentes ao caminho identificados até o momento. Q está estruturado como uma fila de prioridade (*priority queue*) ordenada por custo;

- S é o conjunto de nós visitados;
- P é a lista contendo os nós identificados até o momento como integrantes do menor caminho a partir da origem s;
- EXTRACT$_{MIN}(Q)$ é a operação que exclui do conjunto Q a tupla (l, v, P) contendo o menor custo, retornando o valor da mesma.

Algorithm 1 Pseudo-algoritmo de Dijkstra adaptado

Require: s, d, G

```
1:  Q ← (0, s, NULL)
2:  S ← NULL
3:  while true do
4:      (l, u, P) ← EXTRACT_MIN(Q)
5:      if u ∉ S then
6:          S ← u
7:          if u = d then
8:              return P
9:          end if
10:         P ← u
11:         for each v adjacent to u in G do
12:             if v ∉ S then
13:                 if l[v] > l then
14:                     Q ← (l[v], v, P)
15:                 else
16:                     Q ← (l, v, P)
17:                 end if
18:             end if
19:         end for
20:     end if
21: end while
```

Ensure: P

A seguir, a descrição do comportamento do algoritmo:

1. Linhas 1-2: Inicialização

   O conjunto Q recebe uma tupla contendo o custo *zero*, o nó de origem s e um valor NULO referente ao caminho ainda desconhecido;

2. Linha 3: Busca do menor caminho

   O laço de repetição é iniciado. A condição de parada indica que ele deve ser executado infinitamente. Quando o nó examinado é igual ao nó de destino (linha 7), o algoritmo termina, retornando a lista P que contém o menor caminho do nó de origem s até o nó de destino d;

3. Linha 4:

   As variáveis l, u e P recebem os respectivos valores da tupla de menor custo extraída do conjunto Q;

4. Linhas 5-6:

   Caso o nó u ainda não tenha sido visitado, ele é adicionado ao conjunto S;

5. Linhas 7-9: Fim da execução

   Caso o nó u seja igual ao nó de destino d, a execução do algoritmo é encerrada e a lista P contendo o menor caminho entre s e d retornada;

6. Linha 10:

   O nó u é adicionado à lista P;

7. Linhas 11-19:

   Cada nó v adjacente ao nó u ainda não visitado (linha 12) é então visitado e seu custo $l[v]$ comparado com o custo atual l (linha 13). Caso seja maior, este custo será adicionado ao conjunto Q juntamente com o nó atual v e o caminho atual P (linha 14). Caso seja menor (linha 15), o custo l é devolvido ao conjunto Q juntamente com o nó atual v e o caminho atual P (linha 16);

5.5 Implementação

O Equalize foi concebido como uma suíte de aplicações de rede, montada sobre a plataforma fornecida pelo controlador de rede Beacon [25]. Como um controlador centralizado de rede, o Equalize conta com uma estrutura para armazenar a topologia da rede sob seu controle denominada *base de adjacências*. Essa base mantém em registro as informações

atualizadas sobre todas as adjacências entre os comutadores e o valor de utilização em suas portas. O Equalize também possui uma *tabela MAC* que associa endereços MAC de origem aprendidos ao comutador e a porta onde os pacotes ingressaram na rede, além de um *mapa de caminhos* que guarda todos os caminhos comutador-a-comutador mais curtos.

A proposta apresentada utiliza um mecanismo de atuação no encaminhamento de fluxos baseado em retro alimentação (*feedback*) pela coleta de medições de uso dos enlaces. Para um esquema desse tipo funcionar corretamente, grande atenção deve ser dada à escala de tempo e à sequência de eventos desencadeados na rede.

Com esses objetivos sendo considerados, cada aplicação de rede tem o seu papel na manutenção do equilíbrio do tráfego na rede:

- A aplicação de Descoberta da Topologia cria e envia dinamicamente pacotes de descoberta de camada de rede (*Link Layer Discovery Protocol*s (LLDPs)), agindo proativamente para garantir a descoberta de enlaces entre comutadores adjacentes. Ela também é responsável pelo relato de falhas de conexão nos enlaces, mantendo a base de adjacências atualizada para refletir o estado real da topologia da rede;
- A aplicação Árvore Geradora (*Spanning Tree*) mantém a rede livre de ciclos para os pacotes de difusão (*broadcast*) e *multicast*;
- A aplicação de Coleta de Medições periodicamente solicita, reúne e processa as medições das portas de cada comutador, armazenando na base de adjacências as informações resultantes de utilização das portas;
- A aplicação de Roteamento calcula os caminhos mais curtos (menos utilizados) entre todos os comutadores através do algoritmo de Dijkstra adaptado.
- A aplicação de Aprendizado-Consulta-e-Encaminhamento trata os pacotes que são direcionados para o controlador por não existir ainda uma entrada na tabela dos comutadores associada à alguma ação para tais pacotes. Essa aplicação atua diretamente na concretização do processo de espalhar os fluxos entre os caminhos menos utilizados, através do envio de instruções para adicionar a ação adequada a tal fluxo na tabela dos comutadores pertinentes.

Com a estrutura integrada de aplicações, consegue-se alcançar a conexão e a migração de *hosts* de forma transparente, como se toda a rede fosse uma única rede de camada 2

(equivalente a um enorme comutador). A migração de *hosts* é um processo simples: basta ligar o *host* na nova porta de comutador e, assim que o *host* enviar um ARP gratuito ou outro pacote contendo seu MAC de origem e endereço IP sua nova posição será aprendida, permitindo que a lógica de encaminhamento atue para configurar os caminhos para os fluxos de acordo com a demanda.

Sendo desenvolvido e implantado em um ambiente de RDS, o arcabouço Equalize realiza a comunicação com os comutadores através da interface de gerenciamento tão logo eles estejam ligados, bastando para isso que os mesmos sejam configurados antecipadamente com o endereço de destino do controlador.

A seguir, uma breve descrição de cada uma das aplicações e também de seu funcionamento.

5.5.1 Descoberta da topologia

A aplicação de Descoberta da Topologia mantém a *base de adjacências* atualizada para refletir o estado real da topologia de rede.

Logo após a conexão de novos comutadores à rede e de seu registro com o controlador, a aplicação de Descoberta da Topologia começa a criar e enviar pacotes de prova (LLDPs) para descobrir quais comutadores da rede são adjacentes ao novo comutador e por quais portas seus enlaces são interconectados.

Os pacotes LLDP são enviados periodicamente, assim, além de auxiliar na descoberta de novos enlaces comutador-a-comutador também fornecem uma maneira de se criar um alerta e desencadear ações do controlador para o caso de falhas em que os LLDPs deixam de ser recebidos a partir de algum enlace previamente conhecido e operacional.

Nos experimentos apresentados nessa dissertação, os pacotes LLDP são enviados a cada 15 segundos.

5.5.2 Árvore geradora (*Spanning Tree*)

Enquanto a base de adjacências está sendo povoada, a aplicação Árvore Geradora executa a desativação da capacidade de inundação (*flood*) em todas as portas dos comutadores recém conectados, mudando o estado do bit **no_flood** em cada porta de inativo para ativo. Em seguida, ela calcula uma árvore geradora através da rede e desativa o bit **no_flood** das portas pertencentes à árvore e também das portas conectadas aos *hosts*

permitindo, assim, que os pacotes de difusão (*broadcast*) sejam entregues por estas portas. Desta forma, evita-se ciclos de pacotes de difusão nos instantes seguintes a conexão de um novo comutador. Essa funcionalidade de desabilitar a inundação em todas as portas dos comutadores recém conectados é opcional, mas está habilitada por padrão nessa aplicação.

Portas ligadas a outros comutadores que não pertençam à árvore geradora permanecem com o bit **no_flood** ativo. Isso simplifica o envio de pacotes de difusão (*broadcast*), exigindo apenas uma única ação de *Flood* (inundação) por comutador, ao mesmo tempo em que permite ao controlador lidar com comutadores interconectados por mais de um enlace.

Isso funciona de forma transparente, sem a necessidade de configurar grupos de agregação de enlace (LAGs) ou outros tipos de agrupamentos de enlace (*Link Bundles*) nos elementos da rede.

5.5.3 Coleta de medições e roteamento

A aplicação de Coleta de Medições consulta periodicamente cada comutador sobre as medições de suas portas. Em seguida, a utilização presente em cada porta pertencente a um enlace conectando dois comutadores é calculada e armazenada na base de adjacências. A utilização da porta indica qual é a fração da taxa de transferência em uso pelo comutador transmissor. Ela é calculada considerando-se a relação da taxa média de bits (durante o período de coleta de medições) para a taxa de bits atribuída ao enlace (definida durante a negociação de estabelecimento do enlace).

Nos experimentos apresentados nessa dissertação, a coleta de medições é realizada a cada meio segundo.

A aplicação de Roteamento se encarrega de construir um grafo simplificado, utilizando cada comutador e cada enlace com a menor utilização para alimentar o algoritmo Dijkstra adaptado, que por sua vez realiza o cálculo dos caminhos mais curtos (menos utilizados) entre todos os comutadores. Cada caminho de comutador-a-comutador calculado é armazenado no mapa de caminhos e torna-se disponível para que a função de Encaminhamento possa usá-lo.

Há de se notar que, a fim de simplificar o cálculo do menor caminho quando os comutadores estão ligados por múltiplos enlaces, apenas o enlace menos utilizado é considerado. A base de adjacências cuida internamente para manter a informação de qual enlace é o menos utilizado a cada atualização de medições.

5.5.4 Aprendizado, consulta e encaminhamento

A aplicação de Aprendizado-Consulta-e-Encaminhamento lida com todos os pacotes que o controlador recebe de um comutador resultantes da não existência de uma entrada associada à alguma ação na tabela desse comutador.

Se o endereço MAC de origem desse pacote ainda não é conhecido como originário da porta e do comutador em que ingressou, então ele é aprendido. Em seguida, o MAC de destino é consultado na tabela de MAC. Se o comutador e porta de destino não são conhecidos, todos os comutadores da rede são instruídos para inundar esse pacote em suas portas[2]. Entretanto, se o comutador de destino é o mesmo que o de origem, uma instrução para adicionar o fluxo é enviada à esse comutador juntamente com a liberação desse pacote do *buffer*. Se o comutador de destino é outro que não o de origem, o mapa de caminhos é acessado e os comutadores e portas pertencentes ao caminho menos utilizado são lidos.

Em seguida, a instrução de adição de fluxo é enviada, iniciando o estabelecimento do caminho a partir do comutador de destino até chegar ao comutador de origem e, finalmente, comandar a liberação do pacote do *buffer* do comutador para o seu destino. Dessa forma, evita-se a ausência de uma entrada na tabela do comutador subsequente contendo uma ação adequada associada à esse fluxo e o consequente encaminhamento desse pacote novamente para o controlador ao invés do próximo comutador no caminho.

[2]Lembre-se que, de fato, os pacotes de difusão (*broadcast*) serão inundados para todos os enlaces que estejam conectados aos *hosts*, mas irão seguir pelos caminhos previamente calculados da árvore geradora nos enlaces entre comutadores.

Capítulo 6

Cenário de testes e resultados

Através da distribuição dos fluxos entre os caminhos menos utilizados, espera-se que o rendimento da rede como um todo seja melhorado.

Mas antes da realização dos testes de desempenho, foram realizados testes para validação da característica principal do Equalize: a equalização dos fluxos através do uso dos caminhos menos utilizados.

A intuição é bem simples: medir o tráfego nas portas de conexão entre três comutadores que estão interligados por três enlaces cada, formando uma topologia em anel, sendo dois enlaces com capacidade de dez unidades de tráfego e um com capacidade de cem unidades de tráfego. Três *hosts* foram conectados a cada comutador, sendo que dois *hosts* possuem suas respectivas interfaces de rede com capacidade de dez unidades de tráfego e um *host* com capacidade de cem unidades de tráfego. A Figura 6.1 mostra a rede de validação construída.

A capacidade total entre cada comutador adjacente é de 120 unidades de tráfego.

No teste de validação, foi utilizado o programa *iperf* para geração e medição do tráfego UDP entre os *hosts*. Três *hosts* tem a aplicação iniciada no modo servidor e outros 3 no modo cliente (três outros *hosts* conectados ao comutador intermediário não são utilizados nesse teste, permanecendo inativos). O tráfego é incrementado gradualmente com a geração de um novo fluxo UDP por cada *host* a cada 15 segundos. Cada novo fluxo possui 2 unidades de tráfego e duração de 900 segundos. Assim, cada *host* contribui com tráfego até atingir um valor próximo do limite de sua interface de rede, mais especificamente, um *host* com 100 unidades de capacidade gera até pouco mais de 98 unidades com destino ao seu par de capacidade igual a 100 e os *hosts* com 10 unidades de capacidade geram pouco mais de 8 unidades de tráfego, cada.

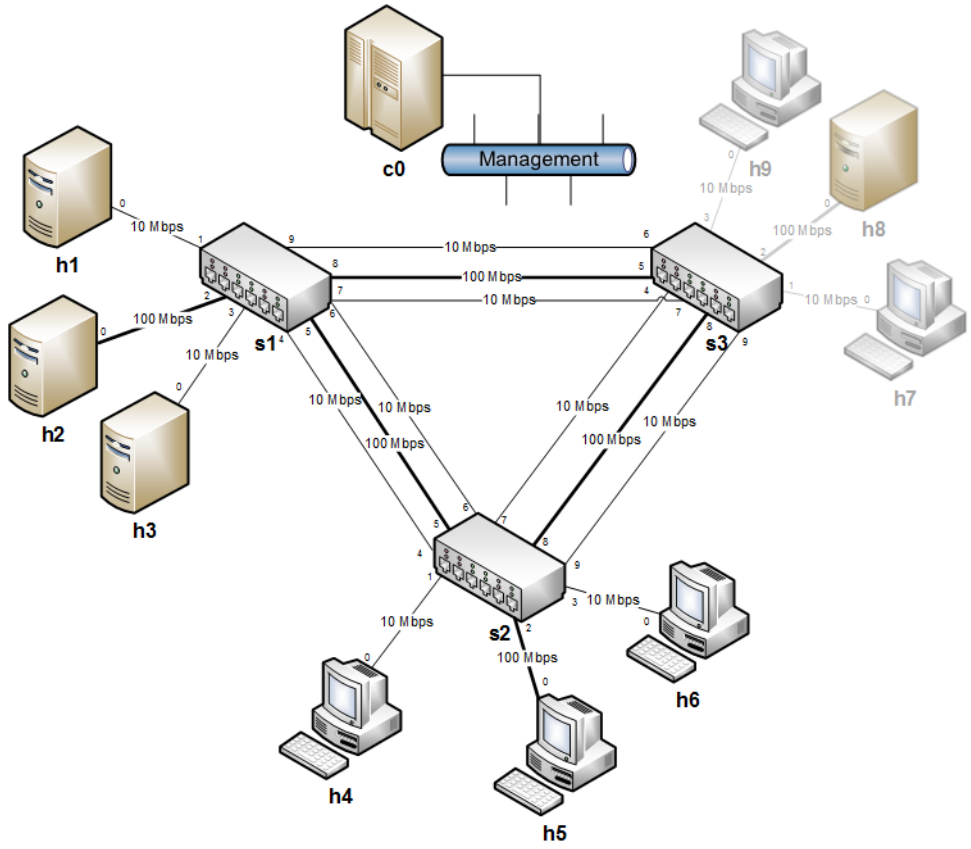

Figura 6.1: Topologia em anel, com múltiplos enlaces de capacidades variadas

A Figura 6.2 mostra o uso equilibrado dos enlaces proporcionado pelo Equalize no teste de validação.

Observa-se que o aumento gradual no tráfego é distribuído de forma homogênea entre as interfaces. Cada duas unidades de tráfego corresponde a 20% e 2% da capacidade das interfaces de 10 e de 100 unidades respectivamente.

Assim, com a característica de balanceamento validada, foi dado início aos testes com o objetivo de comparar o desempenho do arcabouço proposto Equalize com um melhor caso do encaminhamento ECMP.

Esse caso hipotético de ECMP oferece um desempenho de referência para a distribuição de fluxos, pois apenas as perdas por colisão são consideradas. Como todos os endereços MAC de todos os fluxos são conhecidos com antecedência, a tabela de encaminhamento dos comutadores é configurada proativamente na fase inicial dos testes de acordo com um *hash* dos endereços MAC de origem e destino. Assim, as perdas causadas pela distribuição de estado entre os comutadores e escolha dos menores caminhos, além das perdas inerentes ao processo de instalação de novos fluxos quando o protocolo OpenFlow é utilizado de forma reativa são evitadas.

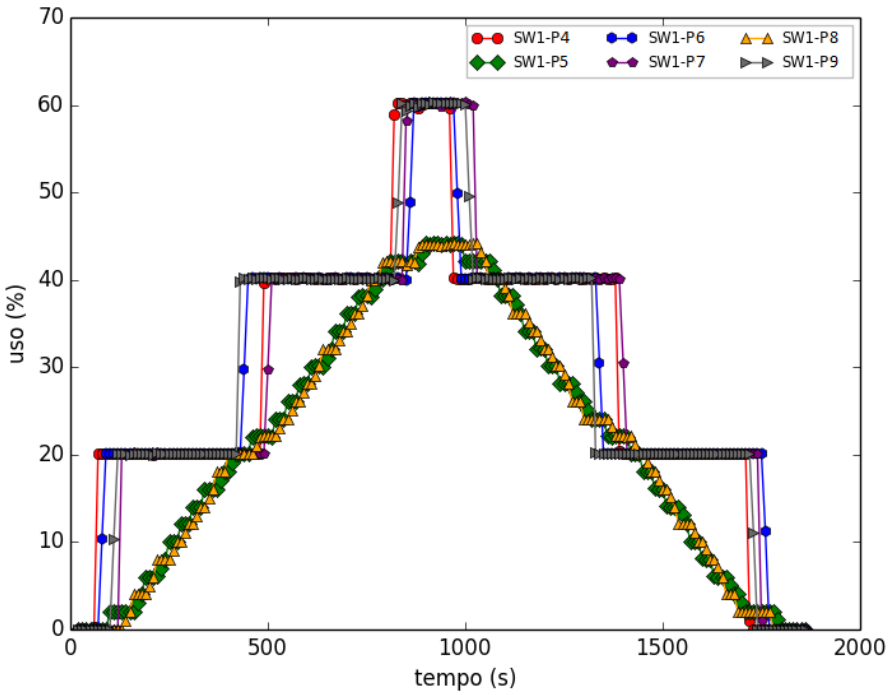

Figura 6.2: Uso equilibrado dos enlaces

Também foram realizados testes considerando apenas um único caminho disponível, através de uma árvore geradora (*spanning tree*). Esse é o pior caso, onde os gargalos ocasionados pela utilização de um único caminho limitam o desempenho da rede, evidenciando as perdas causadas pela falta de utilização dos recursos ociosos.

Por último, foi avaliado como seria o desempenho do tráfego caso todos os *hosts* fossem conectados por um único comutador não bloqueante. Essa é a condição "ideal" para uma rede, ilustrada aqui para mostrar a máxima taxa de transferência alcançável por cada padrão de teste.

Para alimentar a rede, é utilizado um conjunto de padrões de comunicação de acordo com os seguintes estilos:

1. Stride(i): Um *host* com índice x envia para o *host* com o índice $(x+i) mod (num\_hosts)$.

2. Staggered Prob($EdgeP, PodP$): Um *host* envia para outro *host* no mesmo comutador de borda com probabilidade $EdgeP$, e no mesmo *pod* com probabilidade $PodP$, e para o restante da rede com probabilidade $1 - EdgeP - PodP$.

3. Random (aleatório): Um *host* envia para qualquer outro *host* na rede com probabilidade uniforme. Foram incluídos mapeamentos bijetivos e outros onde *hotspots*

estão presentes.

A principal métrica de interesse é a taxa de transferência total alcançada em relação à taxa de transferência total da bissecção da rede. Desta forma, nos experimentos realizados, vinte padrões de comunicação diferentes pertencentes aos estilos mencionados acima são introduzidos na rede usando fluxos TCP para saturar os enlaces.

No primeiro teste de desempenho, todos os *hosts* foram conectados a um único comutador (rede ideal não bloqueante), assim foi observada a máxima taxa de dados alcançada por cada padrão de teste e esses valores foram registrados em relação ao valor da taxa de transferência total da bissecção da rede.

Em seguida, o desempenho do encaminhamento pela árvore geradora foi avaliado. No início do teste a árvore com o menor caminho entre cada *host* é calculada e os comutadores são pré configurados para direcionar os fluxos através dessa árvore.

No teste seguinte, o desempenho do encaminhamento usando o ECMP ideal é medido, com os comutadores estaticamente pré configurados de modo a imitar um *hash* ECMP usando os endereços MAC dos *hosts* de origem e destino.

Por fim, as estatísticas de desempenho durante a execução do controlador Equalize foram avaliadas, e foi validado que o mesmo mantém atualizada a razão de utilização de cada enlace e realiza o encaminhamento dos fluxos de pacotes dinamicamente pelos caminhos menos utilizados.

Os componentes integrantes da configuração dos experimentos realizados são descritos nas próximas Seções e os resultados dos testes discutidos na Seção 6.4.

6.1 Topologia

Para o teste com todos os *hosts* conectados a um único comutador (rede ideal não bloqueante), obviamente a topologia é em Estrela. Para os demais testes, foi utilizada uma simples topologia *fat-tree* com $k = 4$, ilustrada na Figura 6.3.

A topologia *fat-tree* foi escolhida devido às suas características representativas para centros de dados (ver Seção 3.2.3).

Especificamente no teste onde o tráfego flui através de uma árvore geradora *spanning tree* com um único caminho disponível entre os *hosts*, essa sub-topologia foi derivada da topologia *fat-tree* com $k = 4$.

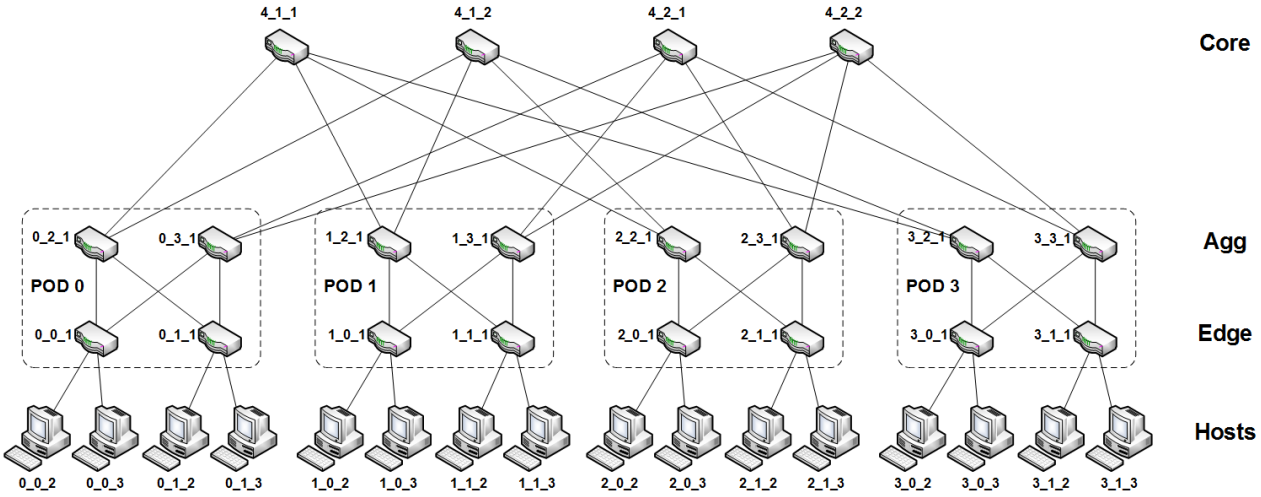

Figura 6.3: Topologia *fat-tree* com $k = 4$ utilizada nos experimentos

6.2 Metodologia

Para uma avaliação rápida e sem dificuldades com *hardware*, os experimentos foram realizados em uma rede virtual criada utilizando o Mininet [54, 33], que apresenta essas características além outras também adequadas à experimentação, conforme já mencionado na Seção 2.6.4.2.

O ambiente de desenvolvimento, implantação e testes utiliza como protocolo de comunicação entre o controlador e os comutadores o OpenFlow 1.0 [59, 2], por esse estar estabelecido a mais tempo e consequentemente ser mais estável além de possuir mais ferramentas com recursos de desenvolvimento e análise de desempenho disponíveis abertamente.

No teste da rede ideal, todos os dezesseis *hosts* foram conectados a um único comutador. Para os demais testes, uma pequena rede *fat-tree* com $k = 4$ (comutadores com 4 portas) contendo dezesseis *hosts* e vinte comutadores como mostrado na Figura 6.3 foi construída, de modo a manter as exigências dos experimentos abaixo do limite de capacidade do *hardware* subjacente.

Os testes experimentais foram executados da seguinte forma: Para os quatro métodos de encaminhamento avaliados separadamente (rede ideal, caminho único (*spanning tree*), ECMP e o Equalize), dezesseis *hosts* criam soquetes para o tráfego de entrada e medem a taxa de transferência de entrada constantemente. Os *hosts*, então, sucessivamente iniciam a transmissão dos seus fluxos, de acordo com os padrões de comunicação descritos na Seção 6.3.

Para os testes com os métodos de encaminhamento de caminho único (*spanning tree*) e ECMP ideal, no passo inicial, o controlador RipL-POX foi utilizado para computar os caminhos e instalar de forma proativa todas as regras de encaminhamento nas tabelas dos comutadores. Nos testes com o Equalize, o intervalo entre as coletas de medições de utilização das portas foi configurado em 500 milissegundos e o cálculo dos menores caminhos usando a média entre a utilização atual e a anterior também foi configurado para ser realizado a cada 500 milissegundos.

A duração do experimento para cada um dos vinte padrões de comunicação é de 60 segundos e utiliza fluxos TCP para saturar os enlaces. A taxa de transferência média da bissecção é observada nos 40 segundos intermediários, ou seja, são desconsiderados os primeiros e os últimos 10 segundos das medições da taxa de transferência de entrada nos *hosts*.

Em seguida, a taxa de transferência total obtida em cada teste é normalizada em relação à taxa máxima da bissecção que poderia ser alcançada na rede em teste, e a média dos resultados de cinco execuções é registrada, juntamente com os valores máximos e mínimos observados.

Os resultados estão consolidados nas Figuras 6.4 a 6.8, e são apresentados e discutidos na Seção 6.4.

6.3 Padrões de teste

Os padrões de comunicação utilizados são os mesmos de [6], e são indicados para estressar e saturar a rede. Tais padrões foram escolhidos pois foram gerados seguindo as linhas de distribuição de tráfego presentes em trabalhos publicados objetivando emular cargas de trabalho típicas de centros de dados. Também estão presentes padrões de comunicação sintéticos passíveis de sobrecarregar as redes de centros de dados.

A seguir, uma breve descrição dos vinte padrões de teste utilizados.

6.3.1 Stride(i)

Foram utilizados quatro padrões de teste do tipo Stride: Stride(1, 2, 4 e 8). Com o aumento dos valores de *stride*, os fluxos atravessam mais camadas (maior número de saltos).

Em Stride(1), o *host* 1 envia tráfego para o *host* 2, o *host* 2 para o *host* 3 e assim por

diante, até o *host* 16 que envia tráfego para o *host* 1.

Em Stride(2), o *host* 1 envia tráfego para o *host* 3, o *host* 2 para o *host* 4 e assim por diante, até o *host* 16 que envia tráfego para o *host* 2.

Em Stride(4), o *host* 1 envia tráfego para o *host* 5, o *host* 2 para o *host* 6 e assim por diante, até o *host* 16 que envia tráfego para o *host* 4.

Finalmente, em Stride(8), o *host* 1 envia tráfego para o *host* 9, o *host* 2 para o *host* 10 e assim por diante, até o *host* 16 que envia tráfego para o *host* 8.

6.3.2 Staggered Prob(EdgeP, PodP)

Foram utilizados seis padrões de teste do tipo Staggered Prob: stag0(0.2,0.3), stag1(0.2,0.3), stag2(0.2,0.3), stag0(0.5,0.3), stag1(0.5,0.3) e stag2(0.5,0.3).

Em stag0, stag1 e stag2(0.2,0.3), um *host* envia tráfego para outro *host* no mesmo comutador de borda com probabilidade de 20%, e para outro *host* no mesmo *pod* com probabilidade de 30%, e para o restante da rede com probabilidade 50%. A diferença entre os três padrões com probabilidade (0.2,0.3) é o *host* de destino.

Já em stag0, stag1 e stag2(0.5,0.3), um *host* envia tráfego para outro *host* no mesmo comutador de borda com probabilidade de 50%, e para outro *host* no mesmo *pod* com probabilidade de 30%, e para o restante da rede com probabilidade 20%. A diferença entre os três padrões com probabilidade (0.5,0.3) é o *host* de destino.

6.3.3 Random

Foram utilizados dez padrões aleatórios (*random*), onde um *host* envia tráfego para qualquer outro *host* na rede com probabilidade uniforme.

Dos dez padrões, três são padrões aleatórios simples (rand0-2), três aleatórios bijetivos (randbij0-2), outros três aleatórios com 2, 3 e 4 fluxos de tráfego por *host* (randx2-4) e, por último, um padrão com *hotspot*.

6.4 Resultados dos testes

As Figuras 6.4 a 6.8 apresentam os resultados da suíte de testes, em uma topologia *fat-tree* com $k = 4$, para os métodos de encaminhamento reativo dinâmico do Equalize, encami-

nhamento proativo estático ECMP, além dos testes com um caminho único, composto por uma topologia em árvore geradora (*spanning tree*) derivada da topologia *fat-tree* representando a forma mais comum de encaminhamento. Também estão incluídos os resultados dos testes com um comutador não bloqueante como uma referência do comportamento de uma rede ideal.

Para cada um dos quatro métodos de encaminhamento analisados, são exibidos os valores de taxa de transferência normalizados (razão da taxa de transferência alcançada pela máxima taxa possível na bissecção da rede). Cada barra representa um dos vinte padrões de comunicação (dentro dos grupos staggered, stride e randomized) indicando o valor médio entre 5 rodadas (com os respectivos máximos e mínimos observados).

As Figuras 6.4 a 6.8 mostram que, conforme esperado, a taxa média alcançada pelo Equalize supera a do caminho único (*spanning tree*) em todos os testes. Porém, observa-se para o esquema de encaminhamento dinâmico reativo usado pelo Equalize a presença de uma variação maior nos valores máximos e mínimos de taxa média de transferência, em comparação aos apresentados pelo encaminhamento estático proativo usado pelo ECMP. Esta variação é atribuída à maior carga imposta ao processamento realizado pelo comutador, que tem de lidar ao mesmo tempo com:

- Pacotes a serem enviados ao controlador por ainda não existir em sua tabela uma entrada associada à alguma ação para eles;
- Instalação de regras de fluxo em suas tabelas;
- Responder ao controlador com as medições solicitadas pelo mesmo.

Tais atividades exigentes de processamento não estão presentes no roteamento proativo, pois neste, os fluxos são conhecidos com antecedência, as regras de fluxo já foram instaladas proativamente nas tabelas durante a inicialização dos testes e o controlador não realiza qualquer solicitação de medições.

Além disso, uma limitação importante inerente à mecânica de funcionamento do protocolo OpenFlow em aplicações reativas (como é o caso do Equalize) é que a taxa de transferência inicial alcançada por um novo fluxo TCP acaba sendo impactada pelo atraso adicional do primeiro pacote. Esse atraso é ocasionado pelo tempo decorrido entre a chegada no controlador da mensagem do comutador informando sobre o primeiro pacote do fluxo e a efetiva instalação de novas regras de encaminhamento na tabela de fluxos dos comutadores no caminho por onde esse novo fluxo deve ser transportado, especialmente

se o tempo total de viagem ida-e-volta (RTT) do canal de gerência entre o comutador e o controlador for maior do que o percebido pelos pacotes do fluxo no caminho de dados, conforme apontado em [18]. Essa limitação também impacta na utilização do gerenciamento centralizado em redes geograficamente distribuídas, deixando o uso do protocolo OpenFlow em aplicações reativas que precisam de alto desempenho limitada a topologias concentradas como as encontradas em centros de dados e campus de universidades.

Para todos os padrões aleatórios e *hotspot*, o Equalize permanece perto o suficiente para ser promissor. Um ajuste fino no tempo de requisição das medições das portas, ou melhor ainda, tendo essas medições processadas e enviadas diretamente e de forma automática como estatísticas de utilização pelos comutadores ao controlador irá proporcionar uma melhora no desempenho do Equalize.

Uma elaboração mais detalhada em torno deste tema é deixada para o Capítulo 8.

A seguir, são apresentados os gráficos com os resultados da suíte de testes para as quatro metodologias de encaminhamento: Caminho único, Equalize e ECMP através de uma topologia *fat-tree* com $k = 4$ e Ponto a ponto (rede ideal) através de um comutador não bloqueante.

6.4.1 Stride(i)

No primeiro teste com o padrão Stride 1, exibido na Figura 6.4, onde cada *host* transmite para o *host* subsequente, nota-se que o Equalize obteve o melhor desempenho, inclusive um pouco acima do alcançado pelo comutador não bloqueante (representativo de uma rede ideal). Esse comportamento deve-se ao fato de que o comutador não bloqueante precisa lidar simultaneamente com o tráfego gerado pelos dezesseis *hosts* na topologia em estrela formada, enquanto que na topologia *fat-tree* a carga em cada comutador é menor, pois, 50% dos *hosts* transmite para o vizinho que está conectado ao mesmo comutador de borda (*edge*), 25% dos *hosts* consegue alcançar o próximo *host* através do comutador de agregação, e o tráfego dos 25% restantes passa pelos comutadores do núcleo da rede.

Já no teste com o padrão Stride 2, o ECMP apresentou um bom desempenho e o Equalize um desempenho intermediário entre ele e o caminhos único. Nota-se que, com o aumento no tráfego cruzando as camadas superiores (agregação e núcleo), todos os três esquemas de roteamento começam a se distanciar do comportamento ideal.

No padrão de teste Stride 4, o desempenho médio do Equalize foi praticamente igual ao do ECMP, apesar da maior variação no resultado entre as rodadas. E, no padrão

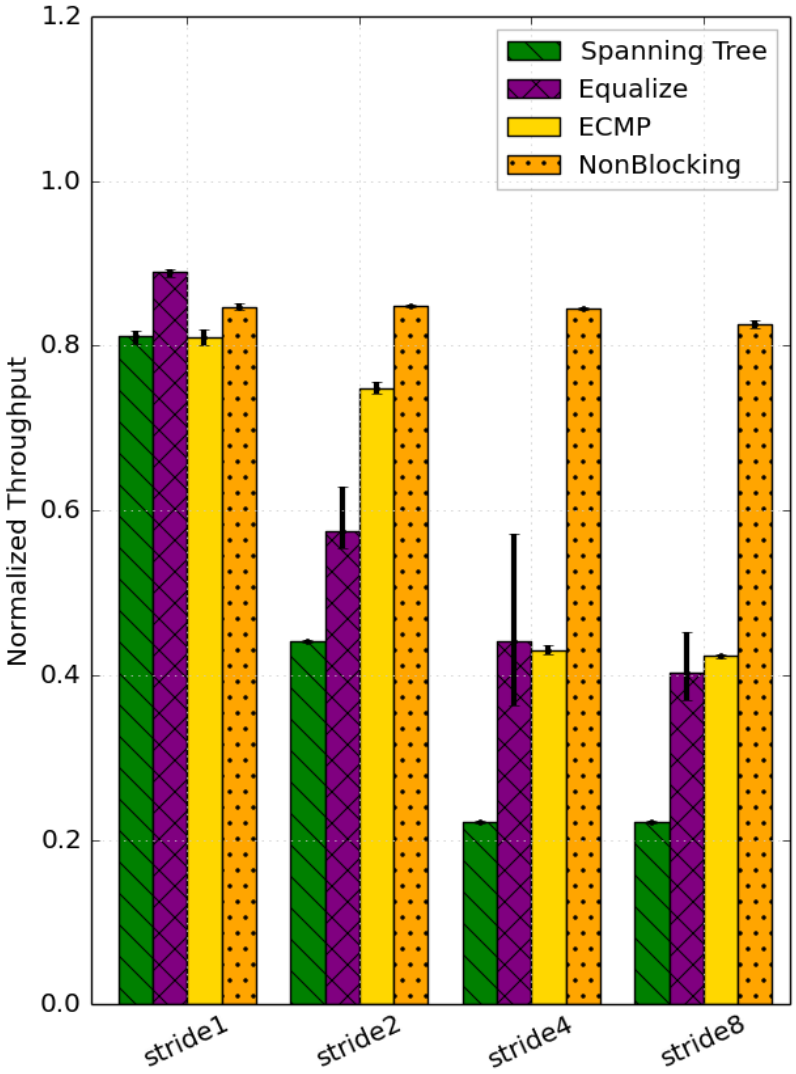

Figura 6.4: Padrões de teste do tipo Stride 1, 2, 4 e 8

de teste Stride 8, o Equalize apresentou um desempenho médio um pouco inferior ao do ECMP.

O tráfego gerado por todos os *hosts*, tanto no Stride 4 quanto no Stride 8, precisa atravessar a camada de comutadores de núcleo para alcançar seu destino. Na topologia de caminho único, o tráfego total alcançado fica em torno de apenas 25% do que a rede disponibiliza, evidenciando o desperdício de recursos que esse método de roteamento proporciona. O ECMP, alcança em torno de 45% em virtude da escolha estática pré estabelecida dos caminhos, causando congestionamento nos comutadores de núcleo. Por fim, o Equalize apresentou uma queda gradativa de desempenho, em virtude, principalmente, do aumento no comprimento do caminho e consequente aumento no número de elementos configurados por fluxo e no tempo de configuração total, impactando no atraso para início efetivo de novos fluxos. Como visto em 5.5.4, o primeiro pacote de um fluxo só é "liberado"

após a configuração de todos os comutadores no caminho da origem até o destino.

O resultado desses testes caracteriza bem o comportamento geral de cada uma das abordagens de roteamento e indica, para o caso do Equalize, que o desempenho alcançado poderá ser melhorado com técnicas para tornar o processamento dos comutadores mais eficiente, tanto no processo de envio de medições (estatísticas) de porta para o controlador quanto no tempo de instalação de fluxos.

Ainda assim, cabe um estudo mais detalhado (a nível "microscópico") do desempenho do ambiente de emulação utilizado, incluindo o Mininet e o Open vSwitch.

6.4.2 Staggered Prob(EdgeP, PodP)

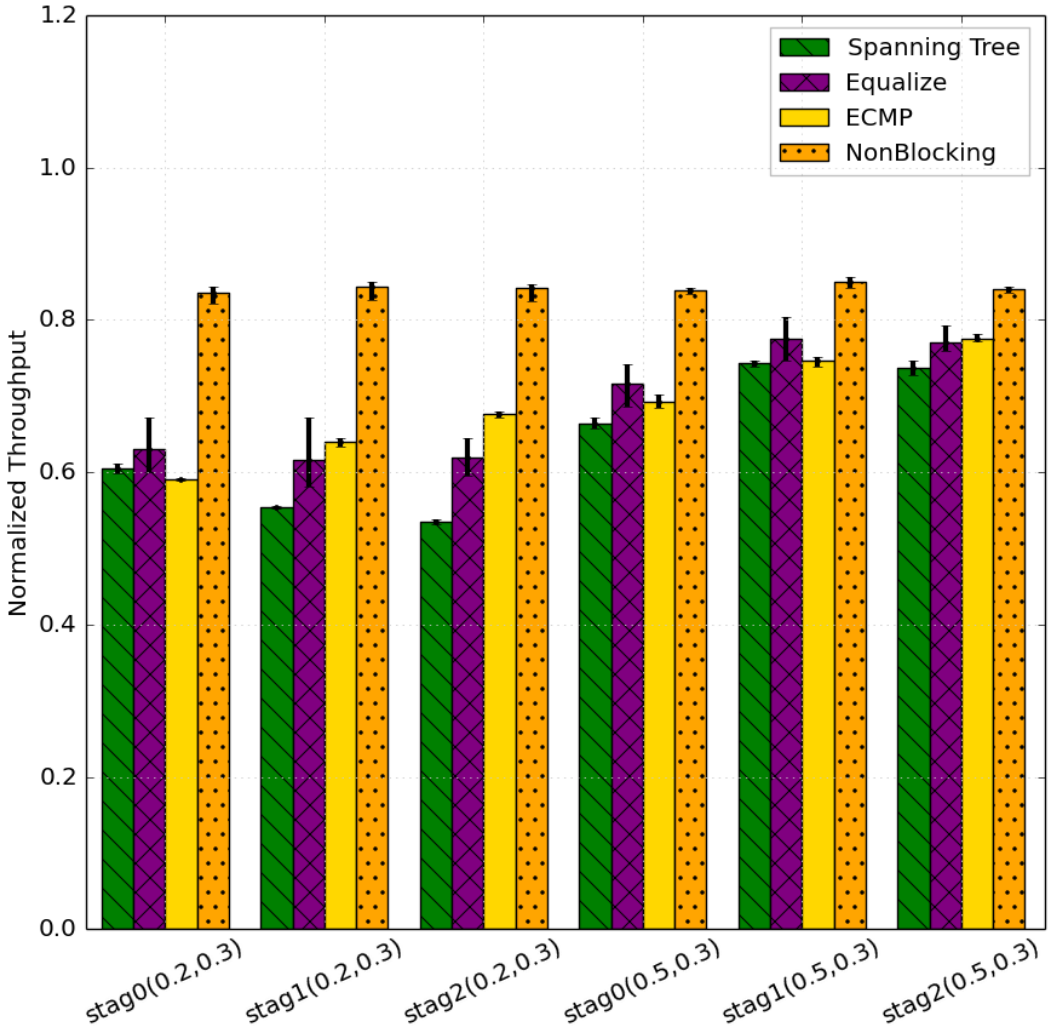

Figura 6.5: Padrões de teste do tipo Staggered Prob(20% Edge, 30% Pod, 50% demais *Hosts*) e Staggered Prob(50% Edge, 30% Pod, 20% demais *Hosts*)

Nos padrões de teste Staggered Prob, o desempenho do Equalize foi similar ao do

ECMP. É notável um melhor desempenho dos três métodos de encaminhamento avaliados (caminho único, Equalize e ECMP) quando a maior parte dos *hosts* se comunica com *hosts* no mesmo comutador de borda (*edge*), como é o caso dos três padrões Staggered Prob(0.5, 0.3).

Já nos três padrões Staggered Prob(0.2, 0.3), onde a probabilidade de comunicação entre *hosts* de diferentes *pods* é maior, os métodos de encaminhamento por caminho único e ECMP apresentaram variação nos resultados com a mudança nos pares de *hosts* se comunicando. Esse comportamento evidencia a natureza estática desses métodos, onde alguns *hosts* acabam tendo como resultado prático um canal de comunicação com capacidade inferior ao obtido por outros *hosts*, dependendo da sua posição em relação aos caminhos predefinidos estaticamente. Em contrapartida, o Equalize apresenta praticamente o mesmo desempenho para esses três padrões.

6.4.3 Random

Os resultados dos testes realizados com os padrões aleatórios (*random*), onde um host envia tráfego para qualquer outro host na rede com probabilidade uniforme, são exibidos nas Figuras 6.6, 6.7 e 6.8.

Nos testes aleatórios simples, alguns *hosts* podem transmitir simultaneamente para outro *host*, daí o motivo pelo qual o comutador não bloqueante apresenta resultados de taxa de transferência mais baixos. O ECMP e o caminho único apresentam um desempenho mais baixo no padrão "rand1". O desempenho do Equalize é inferior ao do ECMP pelos motivos já mencionados no início desta Seção.

Nos testes aleatórios bijetivos, é imposta uma restrição para garantir que cada *host* receba tráfego de apenas um outro *host*. As taxas de transferência alcançadas apresentam melhora em relação aos testes aleatórios simples, pois não há colisões de fluxos na entrada de nenhum *host*. O ECMP exibe aproximadamente a mesma média para as três variantes de tráfego testadas. Mais uma vez, o desempenho do Equalize é inferior ao do ECMP pelos motivos já mencionados no início desta Seção.

Nos três testes com mais de um fluxo por *host*, cada *host* transmite para 2, 3 ou 4 *hosts*. Os três métodos de encaminhamento apresentam melhoria em seu desempenho, conforme aumenta a variedade de fluxos na rede, com redução da taxa individual de cada fluxo proporcional à quantidade de fluxos transmitidos por *host*. Para quatro fluxos, o desempenho do Equalize é similar ao do ECMP.

Para o teste de *hotspot*, também foi aplicada a restrição de um *host* transmitir apenas para um outro. Esse teste estressa os elementos de rede pois os *hosts* transmitem os pacotes enfileirados (*back-to-back*), ou seja, sem intervalo entre eles.

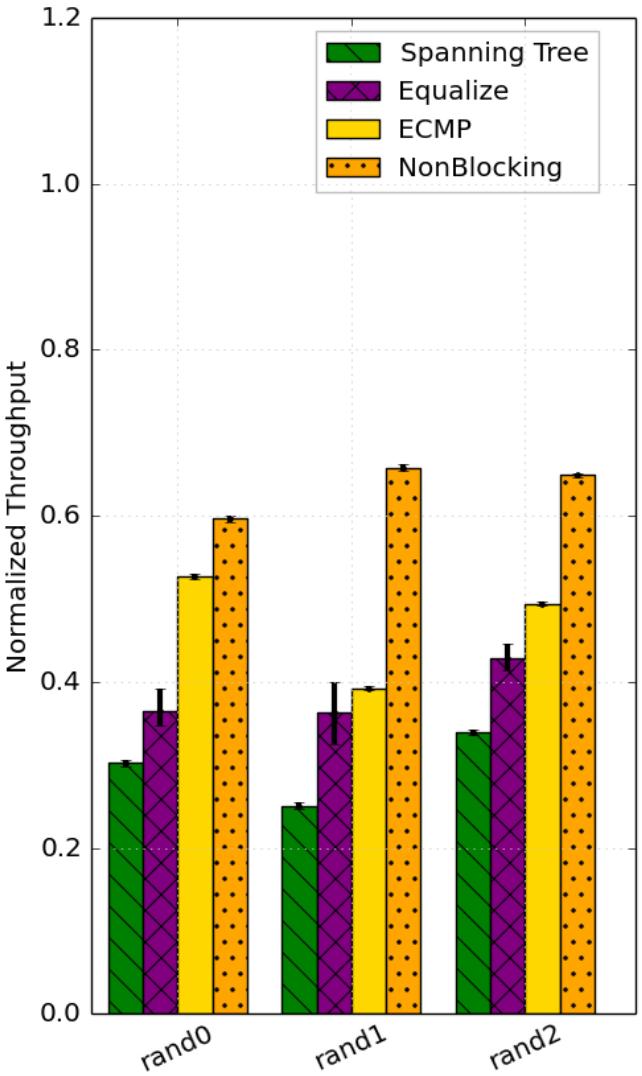

Figura 6.6: Padrões de teste do tipo aleatório simples

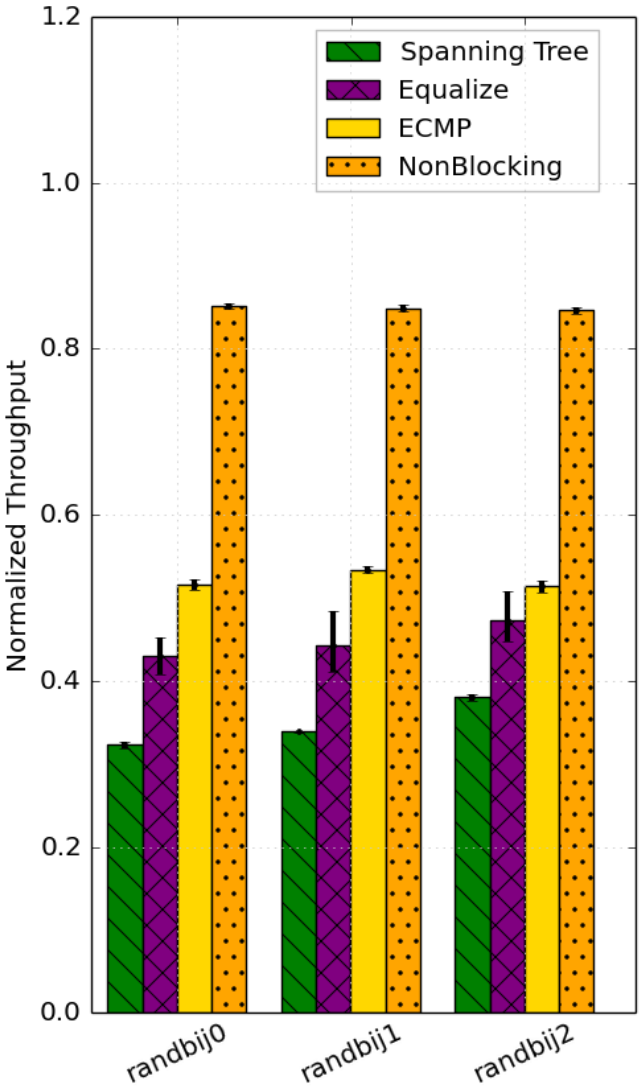

Figura 6.7: Padrões de teste do tipo aleatório bijetivo

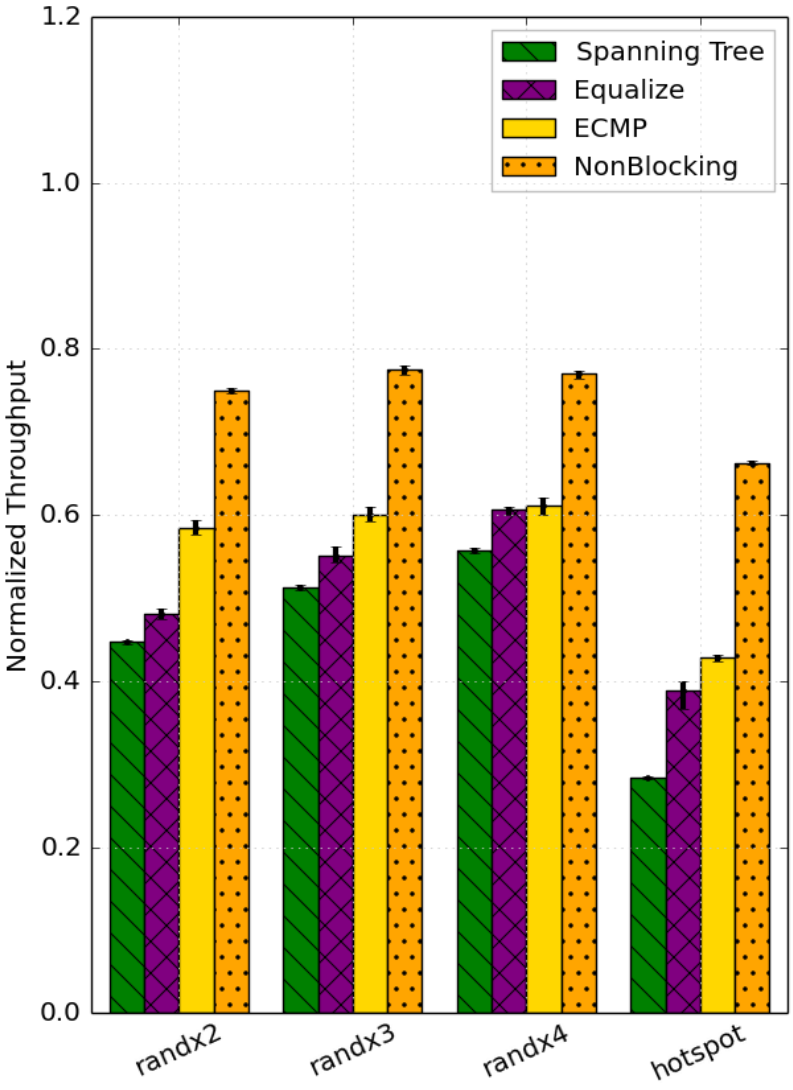

Figura 6.8: Padrões de teste do tipo aleatório com mais de um fluxo por host, e *hotspot*

Capítulo 7

Trabalhos relacionados

Muitas propostas têm surgido com o objetivo de melhorar a eficiência no uso das redes de computadores, explorando uma maior utilização efetiva da taxa de transferência disponível e, em paralelo, o aumento da tolerância a falhas através da utilização dos múltiplos caminhos existentes entre os nós da rede.

Como comentado na Seção 4.3.1, o ECMP apresenta algumas limitações, tais como depender da existência de caminhos "iguais" (com mesmo custo e mesma taxa de transferência), realizar *hash* estático dos fluxos e escolher de forma aleatória os enlaces de destino. Tais limitações acabam ocasionando colisões entre fluxos e o encaminhamento por caminhos mais congestionados.

O arcabouço Equalize apresentado nessa dissertação não se limita a "menores caminhos de mesmo custo" como o ECMP e realiza o encaminhamento inteligente dos fluxos, utilizando o conhecimento global do estado de utilização da rede para a escolha dos melhores caminhos. Em tempo, o Equalize possibilita a distribuição proporcional de fluxos por caminhos de capacidades e comprimentos diferentes.

O Equalize, pode ser utilizado de forma transparente para o encaminhamento de sub fluxos TCP, agregando as vantagens alcançadas pelo MPTCP (ver Seção 4.3.2) ao ganho de equidade realizado pela distribuição homogênea de fluxos por toda a rede.

No âmbito da engenharia de tráfego, especialmente no intra-domínio de operações de Provedores de Serviço da Internet (*Internet Service Providers* - ISPs), Kandula *et al* propõem um protocolo nomeado TeXCP [48] para realizar o balanceamento de carga em tempo real por múltiplos caminhos, de forma a responder dinamicamente às mudanças na demanda de tráfego e ocorrências de falhas ao longo da rede. O TeXCP foi projetado como um protocolo *online* distribuído, que gerencia e move o tráfego adaptativamente dos

caminhos super utilizados para os caminhos subutilizados, de acordo com a demanda dos roteadores de ingresso para os roteadores de egresso da rede. O TeXCP busca reduzir o uso geral dos enlaces da rede através do balanceamento dos fluxos, de forma a estender o tempo de uso da infraestrutura existente antes que o uso máximo de algum enlace exceda um limiar em particular (em torno de 40% de utilização [32]) determinante para que o ISP realize uma atualização (*upgrade*) de sua infraestrutura de rede.

Enquanto TeXCP é voltado para redes de ISPs, que são tradicionalmente superdimensionadas para suportar o crescimento do tráfego ao longo do tempo, o Equalize é voltado para redes de centros de dados onde alcançar efetivamente o uso disponível dos enlaces é crucial para garantir o máximo desempenho e, geralmente, existe sobrescrição nos enlaces das camadas superiores da hierarquia de elementos de rede que são responsáveis pelo encaminhamento dos pacotes. TeXCP utiliza uma abordagem descentralizada, onde cada roteador de ingresso toma a decisão localmente, com base no retorno (*feedback*) recebido dos demais roteadores do núcleo (*core*) da rede. Essa abordagem pode levar à oscilações nos fluxos gerando instabilidade na rede, caso as mudanças de demanda de tráfego entre o par de ingresso-egresso da rede não ocorram em uma escala de tempo muito maior que a dinâmica do balanceamento de carga realizada pelo TeXCP. O Equalize possui controle centralizado, evitando assim o risco de oscilações causadas por decisões locais tomadas pelos nós responsáveis pelo encaminhamento dos fluxos.

A necessidade de aumentar a capacidade de atender requisições de clientes simultaneamente demandou a realização do balanceamento de carga entre os servidores que atendem esses clientes, de forma a alcançar baixa latência nas respostas e alta taxa de transferência. Nas redes legadas, tal balanceamento é alcançado utilizando-se equipamentos específicos (*middleboxes*) dedicados à esta atividade. O paradigma das RDS possibilitou o desenvolvimento de aplicações de rede que transferem essa funcionalidade para os equipamentos da rede responsáveis pelo encaminhamento de pacotes. Pode-se ter apenas um elemento dedicado à essa atividade, mas geralmente, usa-se mais de um, ou ainda todos os elementos da rede para alcançar o balanceamento das requisições dos clientes entre os servidores disponíveis, de forma orquestrada pelo controlador da rede. Koerner e Odej desenvolveram um serviço de balanceamento de carga entre servidores com base em controladores OpenFlow, integrando as funcionalidades de encaminhamento da rede com o balanceamento de carga, de forma a reduzir os esforços de manutenção demandados por soluções proprietárias que envolvem vários servidores conectados em conjunto [50].

A solução proposta nessa dissertação não visa realizar o balanceamento de carga entre

servidores. Ela foi concebida para balancear todo o tráfego presente na rede através de todos os caminhos disponíveis. As abordagens de balanceamento de carga entre servidores como, por exemplo, *Link Balancer Controller* (LBC) [57] e LOBUS [35, 34] serviram de inspiração.

De forma semelhante ao LBC [57], o arcabouço Equalize também cria uma lista de caminhos com base no percentual de utilização dos enlaces calculado a partir de medições do tráfego entre comutadores. O caminho com o menor valor de máxima utilização dentre seus enlaces a ser atribuído ao próximo fluxo ingressante é escolhido a partir desta lista. Então, o estado de encaminhamento adequado é "instalado" na rede física. Também de forma semelhante ao LOBUS [35, 34], o Equalize determina o estado atual da rede, incluindo a topologia da rede e o nível de utilização dos enlaces. Ele também escolhe o caminho com os enlaces menos utilizados, e controla o caminho percorrido pelos pacotes na rede, de modo a equilibrar o tráfego em toda a rede.

Pontos de tráfego intenso (*hotspots*) podem causar exaustão do *buffer* do comutador e, consequentemente, descarte de pacotes que acabam por degradar severamente o desempenho da rede. Para evitar isso, o *Detour-Induced Buffer Sharing* (DIBS) [93] propõe que comutadores congestionados executem um simples desvio aleatório de pacotes quando o *buffer* da porta de saída para o comutador vizinho estiver cheio. Assim, usando a capacidade "extra" disponível dos comutadores próximos, o DIBS alcança uma distribuição de fluxos na rede aproximadamente sem perdas, sem a necessidade de *buffers* adicionais nos comutadores individuais. Com o Equalize, os fluxos são distribuídos de forma justa entre os comutadores, mitigando assim o aparecimento de *hotspots* e, como consequência disso, a capacidade geral da rede em termos de *buffers* é usada de uma forma mais eficiente. Ao evitar a necessidade de ajustar parâmetros de protocolos nos *hosts* e também a ocorrência de ciclos de pacotes inerentes a abordagem utilizada por DIBS, as redes que implementam o arcabouço Equalize são mais robustas e proporcionam uma gerência mais amigável.

Diferente da proposta apresentada como LABERIO [58], que visa mover fluxos que consomem mais recursos de um enlace congestionado para um enlace menos congestionado, a abordagem adotada pelo Equalize é sempre direcionar cada novo fluxo pelo caminho menos utilizado. Se esse novo fluxo chegar a exceder a capacidade máxima de algum enlace ao longo do caminho designado, cabe ao TCP ou a própria aplicação realizar a adaptação em decorrência do descarte de pacotes. Protocolos como o TCP apresentam um melhor desempenho se o caminho ao longo do qual os pacotes são transportados não muda enquanto o fluxo está ativo [40].

Dixit *et al* [22, 23, 21] e Chrysos *et al* [15], defendem que o roteamento por múltiplos caminhos a nível de pacotes entrega a taxa de transferência total de forma consistente para todos os fluxos, permitindo assim uma matriz de comutação de centro de dados eficiente. O Equalize utiliza uma abordagem mais conservadora, considerando a escolha de um mesmo caminho para todos os pacotes pertencentes a um fluxo, de forma a evitar uma redução no desempenho da rede decorrente da degradação causada ao TCP quando um numero alto de pacotes chega ao destino fora de ordem.

Jyothi *et al* [47] defendem o roteamento pela fonte (*source routing*) como uma abordagem simples e escalável para alcançar uma matriz de comutação de rede flexível e de alto desempenho. Quando a rede expõe ao controlador ou aos equipamentos de borda todos os caminhos disponíveis, permite a máxima possibilidade de escolha a esses elementos. Alega-se que, ao codificar e processar de forma compacta em um único campo a rota na fonte, com o OpenFlow 1.3 [66], o desempenho ótimo de vazão é sustentado, além da redução alcançada no tamanho necessário da tabela de encaminhamento. Uma caraterística do Equalize é não necessitar de que seja realizada qualquer alteração nos *hosts*, e a tomada de decisão efetuada apenas pelo controlador simplifica o processo de controle da rede. O Equalize oferece uma estratégia de encaminhamento simples e direcionada para o equilíbrio na utilização dos enlaces da rede.

Capítulo 8

Conclusão e trabalhos futuros

A proposta apresentada nessa dissertação abrange muitas disciplinas e tecnologias de rede, da engenharia de tráfego até o balanceamento de cargas de trabalho e, finalmente, passando pelo gerenciamento de redes. Ao disponibilizar uma visão atualizada do uso dos enlaces da rede, torna-se possível a tomada de decisão objetiva e em tempo real de forma que os fluxos de pacotes encaminhados possam convergir para o uso equilibrado dos enlaces, resultando na melhora global da eficiência na utilização dos recursos da rede.

No entanto, há de se notar que, ao encaminhar pacotes por caminhos com uma maior quantidade de saltos em relação ao mínimo possível, aumenta-se a probabilidade de colisões entre fluxos e esta ação pode acabar resultando em uma qualidade de experiência abaixo de um determinado valor desejado. Ocorre uma troca entre o ganho que poderia ser alcançado em um fluxo individual pelo ganho global em todos os fluxos que trafegam pela rede, gerando como principal benefício uma melhora na eficiência. Ainda assim, pode ser desejável priorizar alguns fluxos em detrimento de outros, realizando seu encaminhamento através dos caminhos com o menor número de saltos.

Esse objetivo pode ser facilmente contemplado pelo arcabouço Equalize, afinal ajustar e incluir algumas linhas de programa é o que se faz em redes definidas por *software*. Tal flexibilidade é impensável na Internet atual. Para o encaminhamento através dos caminhos com o menor número de saltos, basta criar um novo *mapa de caminhos* com o resultado do cálculo usando o algoritmo de Dijkstra padrão alimentado com um custo fixo por elemento, e consultá-lo na fase de encaminhamento condicionado apenas aos fluxos de interesse. Desta forma, os fluxos de interesse são "priorizados" pelos caminhos com o menor número de saltos (com a vantagem de usar o enlace menos congestionado entre dois comutadores interconectados por mais de um enlace), enquanto os demais fluxos são espalhados homogeneamente pelos enlaces menos utilizados da rede. É possível que os

fluxos priorizados acabem criando alguns *hotspots* mas os demais fluxos seguirão preenchendo os demais enlaces disponíveis, convergindo para o equilíbrio no uso dos recursos da rede como um todo.

O conhecimento do estado de cada enlace da rede poderia ser usado para um controle explícito de congestionamento, com o controlador da rede sinalizando diretamente aos *hosts* quando e quanto de tráfego deve ser reduzido. Por outro lado, os *hosts* poderiam solicitar ao controlador a garantia de uma taxa de transferência para alguns de seus fluxos, ou questionar o controlador sobre a capacidade disponível no caminho até outro *host* e ajustar dinamicamente seus fluxos para priorizar a transferência de dados a um *host* com maior capacidade em detrimento de outro cujo caminho esteja mais congestionado.

O balanceamento de carga tradicional entre servidores também pode ser realizado sobre o balanceamento global da rede. Uma aplicação intermediária detectaria os fluxos a serem balanceados entre os servidores, consultaria o *mapa de caminhos* para utilizar os caminhos menos congestionados e finalmente sinalizaria aos comutadores da rede a necessidade de reescrita dos endereços de origem e destino dos fluxos, conforme o caso.

Com o encaminhamento dinâmico realizado e a visão global sobre a utilização da rede alcançados pelo Equalize, comutadores poderiam ser desligados para economizar energia quando a utilização e os padrões de tráfego permitissem e religados caso a demanda aumentasse, de forma similar ao proposto em [39]. Essa atividade é trivial para o caso onde é tolerada interrupção momentânea dos fluxos durante a migração de caminhos e bem mais complexa onde é necessário assegurar que determinados fluxos tenham completado antes da interrupção nos caminhos causada pelo desligamento.

Outra abordagem interessante para aplicação do arcabouço aqui proposto seria em redes onde os *hosts* possuem mais de uma interface conectada, especialmente com o MPTCP sendo utilizado como protocolo de transporte. Uma rede desse tipo é sugerida em [74] e denominada *Dual-homed FatTree* (DHFT). Em uma topologia DHFT, a quantidade de enlaces no nível de acesso é maior do que nos níveis de agregação e núcleo. Se por um lado essa topologia impõe certo grau de sobrescrição para esses níveis, por outro lado ela proporciona benefícios para padrões de tráfego com *hotspots* e padrões onde o núcleo da rede é subutilizado. Além disso, a grande vantagem de se utilizar *hosts* conectados por mais de uma interface à rede é que, caso um enlace entre um *host* e um comutador venha a apresentar falha (seja por causa de uma interface ou cabo com defeito), a conectividade desse *host* será mantida pela outra interface (ainda que a taxa total sofra degradação) e em caso de falha do comutador não ocorrerá isolação de todos os *hosts* conectados aquele

comutador.

8.1 Escalabilidade

Nesta Seção são apresentados alguns aspectos que representam um desafio em termos de escalabilidade e abordagens em direção à sua solução.

O processamento centralizado do estado de rede prontamente levanta a questão da escalabilidade. O desenvolvimento de *hardware* e *software* de redes tem avançado a passos rápidos nos últimos anos e desafios de escalabilidade tem sido superados, ainda que novos desafios tenham surgido, expandindo as fronteiras de flexibilidade e capacidade das redes de computadores. Por exemplo, modificar as entradas das tabelas dos comutadores da rede como um todo é uma tarefa ainda relativamente lenta [46].

Um arcabouço embrionário de rede definida por *software* como apresentado aqui é fácil de ser ajustado a fim de suportar uma rede grande e expansível. O caminho mais prático é fazer uso da distribuição da função de controle através de um aglomerado de servidores, de forma a compartilhar a carga de trabalho mantendo a aplicação de controle como uma entidade logicamente centralizada.

Certamente, o algoritmo de Dijkstra adaptado apresentado nesse trabalho para o cálculo dos caminhos menos congestionados pode ser ainda melhorado, de forma a tirar proveito dos seguintes fatos característicos da abordagem proposta:

- Pelo valor do custo de um enlace ser a razão da taxa de transferência utilizada pela taxa máxima do enlace (utilização), o maior custo ao logo de um caminho simplesmente indica também o custo total daquele caminho. Listar de alguma forma mais eficiente todos os caminhos possíveis, ordenados pelos seus respectivos custos totais (valor do enlace mais utilizado) identificando ao fim do processo o enlace com o menor caminho (primeiro da lista). Deve-se levar em consideração o tempo de cálculo e o espaço ocupado na busca da solução (trabalho total realizado);
- A coleta de medições de cada enlace é realizada periodicamente, sendo que a cada coleta, apenas os caminhos que passem pelo enlace cuja taxa de utilização foi atualizada necessitam ser recalculados;
- Adaptar o trabalho realizado à demanda, seja no cálculo dos menores caminhos ou também na coleta de medições, passando a considerar a frequência de atualização

um fator dependente da taxa média (ou de pico) de chegadas de fluxos por segundo na rede.

Uma tabela apresentando a eficiência de diversos algoritmos criados para solucionar o problema dos menores caminhos entre todos os pares (*All Pairs Shortest Path*) é apresentada em [75]. Outra linha de estudo relaciona-se aos algoritmos de atualização dinâmica dos menores caminhos, onde se busca atualizar os caminhos apenas para os pares afetados pela mudança de custo, ao invés de recalcular os menores caminhos para todos os pares da rede.

Uma questão em aberto é, se limitando o tamanho da topologia através da divisão da rede como um todo em sub grafos tal como definido pela Ponte de Menor Caminho (SPB) [42], irá melhorar a escalabilidade ou aumentar demasiadamente a carga de trabalho no aglomerado de controle.

Um exemplo prático relacionado à escalabilidade foi observado nos testes iniciais do Equalize, com o protótipo criado sobre o controlador POX [1]. Por ser desenvolvido na linguagem Python, que é interpretada e não permite execução paralela, quando o tempo para coleta de medições foi configurado para um valor inferior a 10 segundos, o tempo de processamento das mensagens de chegada de pacotes (*Packet In*) e o envio das mensagens de controle e configuração para os comutadores apresentou um aumento significativo, o que acabou impactando no desempenho geral do controlador. Com esses resultados, foi necessário realizar a busca por uma plataforma de controle mais adequada ao desempenho esperado e, então, o controlador Beacon que suporta processamento paralelo foi escolhido para dar prosseguimento à pesquisa. O controlador Beacon continuou apresentando o desempenho normal no processamento das mensagens de chegada de pacotes (*Packet In*) e no envio das mensagens de controle e configuração para os comutadores, mesmo com o tempo para coleta de medições configurado para apenas 500 milissegundos. A Figura A.1, presente no Apêndice A, mostra os resultados do Equalize executado sobre o POX utilizando um intervalo de 5 segundos para coleta de medições e sobre o Beacon com um intervalo de 500 milissegundos. Como referência, foram incluídos os resultados do caminho único (*spanning tree*) onde a configuração é realizada estaticamente logo no início do experimento.

O Equalize utiliza apenas medições de porta, mais especificamente as medições sobre o total de *bytes* transmitidos pelas portas que pertencem a enlaces com outros comutadores, o que implica em otimização do uso do tempo de processamento dos comutadores em relação ao que seria necessário para uma solução que precisa de uma visão global das

medições de fluxos, como é o caso de Hedera [6], por exemplo. Ainda assim, um mecanismo que pode ser adaptado de forma a beneficiar a escalabilidade do Equalize é proposto em [84]: Um monitor de tráfego assistido por *hardware* que mede a utilização do enlace nos comutadores em velocidade de linha sem afetar seu desempenho de encaminhamento. Com alguns ajustes, cada comutador poderia enviar de forma autônoma ao controlador as estatísticas de uso das portas, juntamente com a estampa de tempo para auxiliar na manutenção do estado de uso da rede.

Uma linha de estudo futuro contempla a análise detalhada (a nível "microscópico") do desempenho do ambiente de emulação utilizado, incluindo o Mininet e o Open vSwitch, com o desenvolvimento e a execução de testes específicos para identificar possíveis limitadores de desempenho e ações de correção e contorno para os mesmos.

Dito isto, pode-se acreditar que o enorme potencial do roteamento baseado no estado de utilização dos enlaces impulsione maiores pesquisas nesta área. Com a evolução da tecnologia, *hardware* e *software*, o desempenho desejado poderá ser alcançado.

8.2 Conclusão

O paradigma das RDS tem facilitado a pesquisa e a experimentação em redes de computadores. As RDS ainda estão em sua infância e existe espaço para a implementação de melhorias tanto nos *softwares* quanto nos *hardwares*. Além dos esforços da comunidade acadêmica empregados na evolução das RDS, os fabricantes de *hardware*, desenvolvedores de *software*, vendedores e integradores de soluções, além de usuários finais como empresas de telecomunicações e de serviços da Internet têm investido em pesquisas e na implantação de RDS em diversas escalas. Soluções comerciais já são encontradas à disposição do mercado e a tendência é o aumento na quantidade de redes híbridas ou totalmente RDS. Seguindo o caminho da evolução dos sistemas operacionais, programas para computadores e dos próprios computadores, espera-se que os comutadores, os controladores de rede e as aplicações de rede nas RDS progridam cada vez de forma mais rápida.

Nessa dissertação foi apresentado o Equalize, um arcabouço concebido para distribuir os fluxos de dados igualmente pela rede, fazendo uso de todos os caminhos disponíveis de forma a alcançar o equilíbrio de utilização na rede e ótima alocação de recursos. Contribuindo, assim, para o uso eficiente dos enlaces e para aumentar o desempenho geral da rede.

Através da comparação entre os resultados obtidos com o Equalize e os obtidos com

outros métodos de encaminhamento, foi mostrado que o Equalize melhora a capacidade de taxa de transferência da rede pela utilização de recursos antes ociosos e desperdiçados. O Equalize oferece um bom desempenho, apesar de ainda haver espaço para a implementação de melhorias.

A metodologia de distribuir o tráfego com base na capacidade utilizada (ou disponível) da rede, transformando-a em auto-balanceada, é promissora e trará como maior benefício o aumento na eficiência do uso da rede.

Por fim, algumas questões para investigação ainda estão abertas. O trabalho em melhorias no controlador implementado irá trazer um aumento nos valores de transferência de tráfego obtidos, contribuindo para melhorar ainda mais a eficiência no uso da rede.

Referências

[1] POX. http://www.noxrepo.org/pox/about-pox/. [Online; acessado 10-Maio-2015].

[2] Openflow 1.0 specification. http://www.openflow.org, 12 2009. [Online; acessado 23-Setembro-2014].

[3] Openflow switching reference system. http://www.openflow.org/wp/downloads/, Dec. 2009. [Online; acessado 10-Maio-2015].

[4] 3rd, D. E. E.; Dutt, D. G.; Gai, S.; Perlman, R.; Ghanwani, A. Routing Bridges (RBridges): Base Protocol Specification. IETF RFC 6325, Oct. 2015.

[5] Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev. 38*, 4 (aug 2008), 63–74.

[6] Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *NSDI* (2010), vol. 10, pp. 19–19.

[7] Alizadeh, M.; Edsall, T. On the data path performance of leaf-spine datacenter fabrics. In *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on* (Aug 2013), pp. 71–74.

[8] Arista. 7320x series data sheet. http://www.arista.com/assets/data/pdf/Datasheets/7320X_DS.pdf, 2015. [Online; acessado 24-Setembro-2015].

[9] Benson, T.; Akella, A.; Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 267–280.

[10] Berman, M.; Chase, J. S.; Landweber, L.; Nakao, A.; Ott, M.; Raychaudhuri, D.; Ricci, R.; Seskar, I. Geni: A federated testbed for innovative network experiments. *Computer Networks 61* (2014), 5 – 23. Special issue on Future Internet Testbeds – Part I.

[11] Bitar, N.; Gringeri, S.; Xia, T. Technologies and protocols for data center and cloud networking. *Communications Magazine, IEEE 51*, 9 (September 2013), 24–31.

[12] Braden, R. Requirements for Internet Hosts - Application and Support. IETF RFC 1123, July 2013.

[13] Braden, R. Requirements for Internet Hosts - Communication Layers. IETF RFC 1122, Mar. 2013.

[14] Casado, M.; Freedman, M. J.; Pettit, J.; Luo, J.; McKeown, N.; Shenker, S. Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev. 37*, 4 (Aug. 2007), 1–12.

[15] Chrysos, N.; Neeser, F.; Gusat, M.; Minkenberg, C.; Denzel, W.; Basso, C. All routes to efficient datacenter fabrics. In *Proceedings of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip* (New York, NY, USA, 2014), INA-OCMC '14, ACM, pp. 41–44.

[16] Cisco. Cisco nx-os software for cisco nexus 7000 series switches. `http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/data_sheet_c78-437306.pdf`, 2015. [Online; acessado 24-Setembro-2015].

[17] Conterato, M.; de Oliveira, I.; Ferreto, T.; De Rose, C. A. Avaliação do suporte à simulação de redes openflow no ns-3. *Anais da 11a. Escola Regional de Redes de Computadores* (2013).

[18] Curtis, A. R.; Mogul, J. C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. Devoflow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 conference* (2011), SIGCOMM '11, pp. 254–265.

[19] de Lima Pinto, L.; Bornstein, C. T. Algoritmos para problemas de caminho Ótimo em grafos. In *XXXVIII SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL* (Goiânia, GO, 2006), SOCIEDADE BRASILEIRA DE PESQUISA OPERACIONAL, pp. 2418–2419.

[20] Di, P.; Wählisch, M.; Wittenburg, G. *Modeling the Network Layer and Routing Protocols*. Springer, 4 2010, ch. 16, pp. 359–384.

[21] Dixit, A.; Prakash, P.; Hu, Y.; Kompella, R. On the impact of packet spraying in data center networks. In *INFOCOM, 2013 Proceedings IEEE* (April 2013), pp. 2130–2138.

[22] Dixit, A.; Prakash, P.; Kompella, R. R. On the efficacy of fine-grained traffic splitting protocols in data center networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (New York, NY, USA, 2011), SIGCOMM '11, ACM, pp. 430–431.

[23] Dixit, A. A.; Prakash, P.; Kompella, R. R.; Hu, C. On the efficacy of fine-grained traffic splitting protocols in data center networks. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2012), SIGMETRICS '12, ACM, pp. 411–412.

[24] Eppstein, D. Spanning trees and spanners. *Handbook of computational geometry* (1999), 425–461.

[25] Erickson, D. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 13–18.

[26] Feamster, N.; Rexford, J.; Zegura, E. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev. 44*, 2 (Apr. 2014), 87–98.

[27] FIBRE. Future internet brazilian environment for experimentation. `http://fibre.org.br`, 2015.

[28] Gill, P.; Jain, N.; Nagappan, N. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference* (New York, NY, USA, 2011), SIGCOMM '11, ACM, pp. 350–361.

[29] Gough, C.; Steiner, I.; Saunders, W. Why data center efficiency matters. In *Energy Efficient Servers*. Apress, 2015, pp. 1–20.

[30] Greenberg, A.; Hamilton, J. R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D. A.; Patel, P.; Sengupta, S. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 51–62.

[31] Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (July 2008), 105–110.

[32] Guichard, J.; Faucheur, F. L.; Vasseur, J.-P. *Definitive MPLS Network Designs*. Cisco Press, 2005.

[33] Handigol, N.; Heller, B.; Jeyakumar, V.; Lantz, B.; McKeown, N. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 253–264.

[34] Handigol, N.; Seetharaman, S.; Flajslik, M.; Gember, A.; McKeown, N.; Parulkar, G.; Akella, A.; Feamster, N.; Clark, R.; Krishnamurthy, A., et al. Aster* x: Load-balancing web traffic over wide-area networks, 2009.

[35] Handigol, N.; Seetharaman, S.; Flajslik, M.; McKeown, N.; Johari, R. Plug-n-serve: Load-balancing web traffic using openflow.

[36] Heller, B. Ripcord-lite for pox: A simple network controller for openflow-based data centers. `https://github.com/brandonheller/riplpox.`, Apr. 2012. [Online; acessado 10-Maio-2015].

[37] Heller, B.; Erickson, D.; McKeown, N.; Griffith, R.; Ganichev, I.; Whyte, S.; Zarifis, K.; Moon, D.; Shenker, S.; Stuart, S. Ripcord: A modular platform for data center networking. In *Proceedings of the ACM SIGCOMM 2010 Conference* (New York, NY, USA, 2010), SIGCOMM '10, ACM, pp. 457–458.

[38] Heller, B.; Erickson, D.; McKeown, N.; Griffith, R.; Ganichev, I.; Whyte, S.; Zarifis, K.; Moon, D.; Shenker, S.; Stuart, S. Ripcord: a modular platform for data center networking. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2010), –.

[39] Heller, B.; Seetharaman, S.; Mahadevan, P.; Yiakoumis, Y.; Sharma, P.; Banerjee, S.; McKeown, N. Elastictree: Saving energy in data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 17–17.

[40] Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. IETF RFC 2992, Mar. 2000.

[41] IEEE. 802.1d - mac bridges. `http://www.ieee802.org/1/pages/802.1D.html`, 2004. [Online; acessado 07-Março-2015].

[42] IEEE. 802.1aq - shortest path bridging. `http://www.ieee802.org/1/pages/802.1aq.html`, 2012. [Online; acessado 23-Setembro-2014].

[43] IETF. Transparent interconnection of lots of links (trill) wg. `http://www.ietf.org/html.charters/trill-charter.html`, 2005. [Online; acessado 07-Março-2015].

[44] Iyengar, J.; Raiciu, C.; Barre, S.; Handley, M. J.; Ford, A. Architectural Guidelines for Multipath TCP Development. IETF RFC 6182, Oct. 2015.

[45] Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; Zolla, J.; Hölzle, U.; Stuart, S.; Vahdat, A. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 3–14.

[46] Jin, X.; Liu, H. H.; Gandhi, R.; Kandula, S.; Mahajan, R.; Zhang, M.; Rexford, J.; Wattenhofer, R. Dynamic scheduling of network updates. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 539–550.

[47] Jyothi, S. A.; Dong, M.; Godfrey, P. B. Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (New York, NY, USA, 2015), SOSR '15, ACM, pp. 10:1–10:8.

[48] Kandula, S.; Katabi, D.; Davie, B.; Charny, A. Walking the tightrope: Responsive yet stable traffic engineering. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2005), SIGCOMM '05, ACM, pp. 253–264.

[49] Kobayashi, M.; Seetharaman, S.; Parulkar, G.; Appenzeller, G.; Little, J.; van Reijendam, J.; Weissmann, P.; McKeown, N. Maturing of openflow and software-defined networking through deployments. *Computer Networks 61* (2014), 151 – 175. Special issue on Future Internet Testbeds – Part I.

[50] Koerner, M.; Kao, O. Multiple service load-balancing with openflow. In *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on* (2012), pp. 210–214.

[51] Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T.; Shenker, S. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.

[52] Kreutz, D.; Ramos, F.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE 103*, 1 (Jan 2015), 14–76.

[53] Lantz, B. Mininet. `https://github.com/mininet`, 1999. [Online; acessado 07-Março-2015].

[54] 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, 2010), Hotnets-IX, ACM, pp. 19:1–19:6.

[55] Leiserson, C. E. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers C-34*, 10 (Oct 1985), 892–901.

[56] Levin, D.; Canini, M.; Schmid, S.; Schaffert, F.; Feldmann, A. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 333–346.

[57] Levin, D.; Wundsam, A.; Heller, B.; Handigol, N.; Feldmann, A. Logically centralized?: State distribution trade-offs in software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 1–6.

[58] Long, H.; Shen, Y.; Guo, M.; Tang, F. Laberio: Dynamic load-balanced routing in openflow-enabled networks. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on* (2013), pp. 290–297.

[59] McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[60] Moore, J. T.; Nettles, S. M. Towards practical programmable packets. In *Proceedings of the 20th Conference on Computer Communications (INFOCOM). Citeseer* (2001), Citeseer.

[61] Moreira, M. D.; Fernandes, N. C.; Costa, L.; Duarte, O. Internet do futuro: Um novo horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2009* (2009), 1–59.

[62] Moy, J. T. OSPF Version 2. IETF RFC 2328, Mar. 2013.

[63] Niranjan Mysore, R.; Pamboris, A.; Farrington, N.; Huang, N.; Miri, P.; Radhakrishnan, S.; Subramanya, V.; Vahdat, A. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev. 39*, 4 (Aug. 2009), 39–50.

[64] NSAM. NS-3. https://www.nsnam.org/, 2015. [Online; acessado 24-Setembro-2015].

[65] Nunes, B.; Mendonca, M.; Nguyen, X.-N.; Obraczka, K.; Turletti, T. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE 16*, 3 (Third 2014), 1617–1634.

[66] ONF. Openflow 1.3 specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf, 6 2012. [Online; acessado 23-Setembro-2014].

[67] ONF. Openflow 1.5 specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf, 12 2014. [Online; acessado 10-Maio-2015].

[68] ONF Market Education Committee. Software-defined networking: The new norm for networks. https://www.opennetworking.org, Apr. 2012. [Online; acessado 10-Maio-2015].

[69] Onur, S.; Hallyn, S.; Graber, S.; Andersen, T. LXC - Linux Containers . https://linuxcontainers.org/, 2015.

[70] Perlman, R. An algorithm for distributed computation of a spanningtree in an extended lan. In *Proceedings of the Ninth Symposium on Data Communications* (New York, NY, USA, 1985), SIGCOMM '85, ACM, pp. 44–53.

[71] Perlman, R.; Touch, D. J. D. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. IETF RFC 5556, Oct. 2015.

[72] Peterson, L. L.; Davie, B. S. *Computer Networks, Fifth Edition: A Systems Approach*, 5th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[73] Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; Amidon, K.; Casado, M. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, 2015), USENIX Association, pp. 117–130.

[74] Raiciu, C.; Barre, S.; Pluntke, C.; Greenhalgh, A.; Wischik, D.; Handley, M. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 Conference* (New York, NY, USA, 2011), SIGCOMM '11, ACM, pp. 266–277.

[75] Schrijver, A. *Combinatorial optimization: polyhedra and efficiency*, vol. 24. Springer Science & Business Media, 2003.

[76] Sherry, J.; Ratnasamy, S. A survey of enterprise middlebox deployments. Tech. Rep. UCB/EECS-2012-24, EECS Department, University of California, Berkeley, Feb 2012.

[77] Sherwood, R.; Gibb, G.; Yap, K.-K.; Appenzeller, G.; Casado, M.; McKeown, N.; Parulkar, G. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep* (2009), 1–13.

[78] Singla, A.; Hong, C.-Y.; Popa, L.; Godfrey, P. B. Jellyfish: Networking data centers randomly. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (San Jose, CA, 2012), USENIX, pp. 225–238.

[79] Stringer, J.; Pemberton, D.; Fu, Q.; Lorier, C.; Nelson, R.; Bailey, J.; Correa, C.; Esteve Rothenberg, C. Cardigan: Sdn distributed routing fabric going live at an internet exchange. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (June 2014), pp. 1–7.

[80] Tanenbaum, A. *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[81] Tennenhouse, D. L.; Smith, J. M.; Sincoskie, W. D.; Wetherall, D. J.; Minden, G. J. A survey of active network research. *Comm. Mag. 35*, 1 (Jan. 1997), 80–86.

[82] Tennenhouse, D. L.; Wetherall, D. J. Toward an active network architecture, 1996.

[83] Thaler, D.; Hopps, C. Multipath Issues in Unicast and Multicast Next-Hop Selection. IETF RFC 2991, Mar. 2000.

[84] Tso, F. P.; Hamilton, G.; Weber, R.; Perkins, C.; Pezaros, D. Longer is better: Exploiting path diversity in data center networks. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on* (July 2013), pp. 430–439.

[85] Wikipédia. Função hash. `https://pt.wikipedia.org/wiki/Funç~ao_hash`, 2015. [Online; acessado 24-Setembro-2015].

[86] Wikipédia. Grafo orientado. `https://pt.wikipedia.org/wiki/Grafo_orientado`, 2015. [Online; acessado 26-Setembro-2015].

[87] Wikipédia. Internet protocol suite. `https://en.wikipedia.org/wiki/Internet_protocol_suite`, 2015. [Online; acessado 26-Setembro-2015].

[88] Wikipédia. Minimum spanning tree. `https://en.wikipedia.org/wiki/Minimum_spanning_tree`, 2015. [Online; acessado 26-Setembro-2015].

[89] Wikipédia. Modelo OSI. `https://pt.wikipedia.org/wiki/Modelo_OSI`, 2015. [Online; acessado 26-Setembro-2015].

[90] Wikipédia. Rede de computadores. `https://pt.wikipedia.org/wiki/Rede_de_computadores`, 2015. [Online; acessado 26-Setembro-2015].

[91] Wikipédia. TCP/IP. `https://pt.wikipedia.org/wiki/TCP/IP`, 2015. [Online; acessado 26-Setembro-2015].

[92] Wu, X.; Yang, X. Dard: Distributed adaptive routing for datacenter networks. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on* (June 2012), pp. 32–41.

[93] Zarifis, K.; Miao, R.; Calder, M.; Katz-Bassett, E.; Yu, M.; Padhye, J. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 6:1–6:14.

APÊNDICE A – Desempenho do Equalize no POX

Durante o desenvolvimento inicial do Equalize, foi utilizado o controlador POX como base. Depois, para um melhor desempenho, foi utilizado o controlador Beacon.

Abaixo uma comparação do desempenho do Equalize executado sobre o POX e sobre o Beacon, e também o desempenho do encaminhamento estático por caminho único como referência.

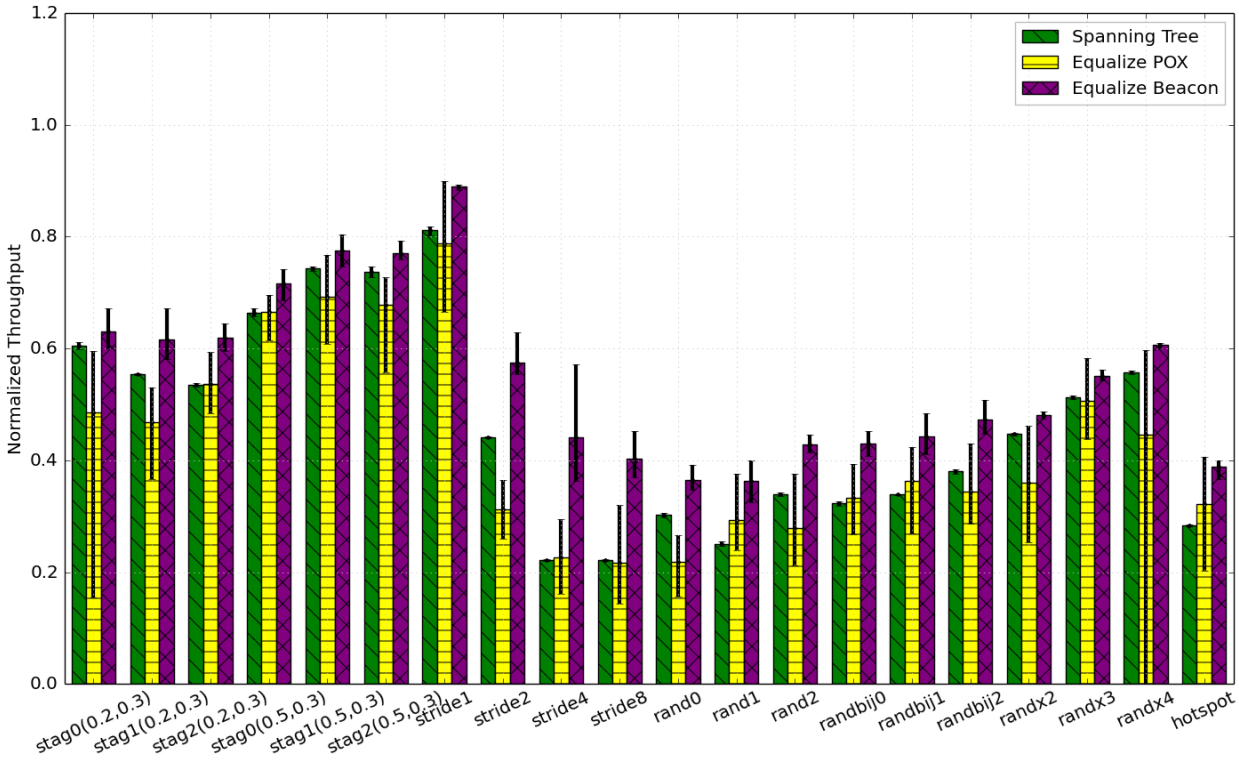

Figura A.1: Comparação entre o desempenho do Equalize executado sobre o POX e sobre o Beacon. Como referência, o desempenho do encaminhamento estático por caminho único (*spanning tree*).