MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

NATÁLIA QUEIROZ DE OLIVEIRA

EMPREGO DE SDN PARA O BALANCEAMENTO DE CARGA EM REDES DE COMPUTADORES COM SUPORTE A MÚLTIPLOS CAMINHOS

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

NATÁLIA QUEIROZ DE OLIVEIRA

EMPREGO DE SDN PARA O BALANCEAMENTO DE CARGA EM REDES DE COMPUTADORES COM SUPORTE A MÚLTIPLOS CAMINHOS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Ten Cel Ronaldo Moreira Salles - Ph.D.

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

004.6 Oliveira, Natália Queiroz de

048e Emprego de SDN para o Balanceamento de Carga em Redes de Computadores com Suporte a Múltiplos Caminhos/ Natália Queiroz de Oliveira, orientada por Ronaldo Moreira Salles – Rio de Janeiro: Instituto Militar de Engenharia, 2014.

102 p.: il.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2014.

1. Sistemas e Computação - teses e dissertações. 2. Arquitetura e Redes de Computadores - SDN. I. Salles, Ronaldo Moreira. II. Título. III. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

NATÁLIA QUEIROZ DE OLIVEIRA

EMPREGO DE SDN PARA O BALANCEAMENTO DE CARGA EM REDES DE COMPUTADORES COM SUPORTE A MÚLTIPLOS CAMINHOS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Ten Cel Ronaldo Moreira Salles - Ph.D.

Aprovada em 27 de Agosto de 2014 pela seguinte Banca Examinadora:

Ten Cel Ronaldo Moreira Salles - Ph.D. do IME - Presidente

Prof. Ricardo Choren Noya - D.Sc. do IME

Prof. Sidney Cunha de Lucena - D.Sc. da UNIRIO

Rio de Janeiro
2014

À meus amados pais

AGRADECIMENTOS

Primeiramente agradeço a Deus pela minha vida, e por ter me dado força para vencer esse desafio.

Aos meus pais, Adauto e Maria de Fátima que sempre me apoiaram em todos os momentos da minha vida.

Ao meu noivo Alex Ferreira por todo apoio, amor e paciência.

Ao meu orientador, Ronaldo Moreira Salles, por suas disponibilidades e principalmente pelas relevantes observações realizadas ao longo deste trabalho.

A todas as pessoas que contribuiram com o desenvolvimento desta dissertação de mestrado, tenha sido por meio de críticas, ideias, apoio, incentivo ou qualquer outra forma de auxílio.

E por fim, por todo conhecimento e ensinamentos transmitidos, agradeço a todos os professores, funcionários e alunos do Departamento de Engenharia de Sistemas (SE/8) do Instituto Militar de Engenharia.

Muito Obrigada!!!

Natália Queiroz de Oliveira

"No meio da dificuldade encontra-se a oportunidade."

Albert Einstein

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

ABREVIATURAS

AFLCMF	-	*Adaptive Flow-Level Load Control Scheme for Multipath Forwarding*
API	-	*Application Programming Interface*
BGP	-	*Border Gateway Protocol*
DH	-	*Direct Hashing*
DPID	-	*Datapath ID*
ECMP	-	*Equal-cost multi-path routing*
EDPF	-	*Earliest Delivery Path First*
FAQ	-	*Frequently Asked Questions*
FLARE	-	*Flowlet Aware Routing Engine*
FS	-	*Fast Switching*
GB	-	*GigaByte*
GUI	-	*Graphical user interface*
HRW	-	*Highest Random Weight*
HT	-	*Hash Threshold*
ID	-	*Identify*
IGP	-	*Interior Gateway Protocol*
IGRP	-	*Interior Gateway Routing Protocol*
IP	-	*Internet Protocol*
JAR	-	*Java ARchive*
LAN	-	*Local Area Network*
LBPF	-	*Load Balancing for Parallel Forwarding*
LDM	-	*Load Distribution over Multipath*
MAC	-	*Media Access Control*
MBD-/ADBR	-	*Progressive Multiple Bin Disconnection with Absolute Difference Bin Reconnection*
MPLS	-	*Multiprotocol Label Switching*
OSPF	-	*Open Shortest Path First*
PBP-RR	-	*Packet-By-Packet Round-Robin*
PMN-LB	-	*Primary Number Modulo-N Load Balance*

PWFR	-	*Packet-By-Packet Weighted Fair Routing*
QoS	-	*Quality of Service*
RAM	-	*Random access memory*
RIP	-	*Routing Information Protocol*
SDN	-	*Software Defined Networking*
SGBD	-	*Sistema Gerenciador de Banco de Dados*
SO	-	*Sistema Operacional*
SQL	-	*Structured Query Language*
SRR	-	*Surplus Round Robin*
TCP	-	*Transmission Control Protocol*
TH	-	*Table-Based Hashing*
THR	-	*Table-Based Hashing with Reassignment*
ToS	-	*Type of Service*
TS-EDPF	-	*Time-Slotted Earliest Delivery Path First*
UDP	-	*User Datagram Protocol*
WRR	-	*Weighted Round Robin*
WIRR	-	*Weighted Interleaved Round Robin*

RESUMO

A atual arquitetura das redes de computadores é formada de protocolos projetados para serem definidos de maneira isolada, provendo soluções específicas. Isso resultou em uma das principais limitações das redes atuais: a complexidade. Os métodos de balanceamento de carga são em sua maioria complexos de serem implementados nas redes atuais, pois os algoritmos e soluções propostas são limitadas pelos fabricantes e disponíveis em hardwares proprietários. As arquiteturas de redes existentes não foram projetadas para atender os requisitos dos usuários atuais, tornando assim a rede limitada, sem espaço para inovação. Surge então uma grande necessidade de gerenciar toda essa demanda de forma ágil e segura.

Redes definidas por *software* conhecida como SDN, acrônimo do inglês, (*Software Defined Networking*) é uma tecnologia emergente onde o plano de encaminhamento e o plano de controle estão claramente separados. O plano de controle pode ser diretamente programado a nível de aplicação. Em SDN, a inteligência da rede é centralizada em controladores, os quais mantém uma visão global da rede. OpenFlow é o protocolo e o elemento essencial para o conceito de SDN. Atualmente, é o único protocolo padronizado que permite a manipulação direta do plano de encaminhamento dos dispositivos de rede.

O presente trabalho propõe uma arquitetura para balanceamento de fluxos, chamada LBX, a qual define o caminho de custo mínimo, baseada no algoritmo Dijkstra, em uma topologia com suporte a múltiplos caminhos. Para a implementação da proposta foi utilizado o controlador Floodlight e o emulador de redes Mininet. Os resultados dos experimentos mostram que o balanceador de fluxos LBX oferece vantagens comparado com os testes sem o balanceamento de carga e com a solução ECMP (*Equal Cost Multipath Routing*) implementado nos roteadores da empresa Cisco.

ABSTRACT

The current architecture of computer networks consists of protocols designed to be defined in isolation, providing specific solutions. This resulted in a major limitation of current computer networks: complexity. The methods of load balancing are complex to be implemented in actual networks mostly because the algorithms and proposed solutions are limited by manufacturers and available in proprietary hardware. The existing network architectures were not designed to meet the requirements of current users, thus making the network limited, with no space for innovation. Then comes the need to manage all this demand agile and safe.

Software defined networking, known as SDN, the English acronym (Software Defined Networking) is an, emerging technology where the forwarding plane and the control plane are clearly separated. The control plane can be directly programmed on the application level. In SDN, network intelligence is centralized in controllers, which keeps a global view of the network. OpenFlow is the essential element on the concept of SDN and is currently the only standardized protocol that allows direct manipulation of the forwarding plane of the network devices.

This paper proposes an architecture for balacing flows, called LBX, which defines the shortest path based on Dijkstra algorithm in a topology with support for multiple paths. To implement the proposal, the Floodlight controller and networks emulator Mininet were used. The experimental results show that the balancer flows LBX offers advantages compared with tests without load balancing and the ECMP (Equal Cost Multipath Routing) solution implemented on Cisco enterprise.

1 INTRODUÇÃO

A explosão de dispositivos móveis, virtualização de servidores e serviços em nuvem estão entre fortes tendências que guiam a indústria de redes a reavaliar as arquiteturas das redes tradicionais. As arquiteturas de redes existentes não foram projetadas para suportar os novos requerimentos dos usuários atuais. O constante crescimento das redes de computadores provocou muitos problemas de desempenho, congestionamento e interrupção de serviços ocasionados por uma sobrecarga normal dos sistemas. A solução mais utilizada para amenizar esses problemas é o balanceamento de carga. No balanceamento da utilização da rede, o tráfego é reencaminhado por vários caminhos alternativos a fim de descongestionar os recursos da rede. A necessidade de balanceamento de carga é uma das principais questões que atraem grande quantidade de pesquisa e uma série de abordagens têm sido propostas (WANG, 1999), (ROY, 2008), (PRABHAVAT, 2011) e (SINGH, 2012).

De acordo com (AWDUCHE, 1999), o congestionamento dos recursos da rede deriva de duas principais causas: recursos inadequados da rede e distribuição de tráfego não balanceado. Os recursos inadequados podem ser melhorados a partir da adição de mais capacidade (em longo prazo) ou da aplicação de mecanismos de controle de tráfego, como controle de admissão, para que assim, tais recursos possam estar adequados às necessidades da rede. A distribuição de tráfego não balanceado é devido aos fluxos de entrada não serem propriamente distribuídos para os recursos disponíveis na rede, levando assim, ao congestionamento de alguns recursos enquanto outros são sub-utilizados. Esse descompasso ocorre como resultado do crescimento do tráfego, bem como das funcionalidades e capacidades limitadas das tecnologias IPs convencionais. Na Internet atual, o roteamento intra domínio é realizado por algoritmos que calculam o caminho de custo mínimo para um determinado destino (SIRIPONGWUTIKORN, 2002).

O roteamento é um processo de encaminhamento de pacotes dentro de uma rede de computadores. A maioria dos dispositivos de redes são capazes de descobrir rotas entre as redes e armazenar suas informações em tabelas de roteamento. A presença de várias interfaces físicas e lógicas incorporadas a um protocolo de roteamento, que utilize o conceito de roteamento de múltiplos caminhos, permite que os recursos da rede possam utilizar

vários caminhos no estabelecimento de conexões simultâneas. Com isso, temos várias possibilidades de seleção de caminho para o tráfego das informações nas redes com suporte a múltiplos caminhos. Existem várias propostas de algoritmos e métodos para seleção de caminho e balanceamento de carga, porém a maioria é baseada em soluções fechadas e padronizadas por aplicações proprietárias. Tais *APIs* não possibilitam a personalização dos seus recursos e funcionalidades, tornando-se complexas de serem implementadas e sem espaço para inovação. A falta de ambientes que permitam a validação desses algoritmos é uma das grandes dificuldades para a adoção de propostas que solucionam estas questões, pois é difícil reproduzir as mesmas características e volumes de tráfego real. Com isso, ficamos limitados a avaliar as soluções já propostas a fim de entendermos melhor o problema e propormos melhorias. Sendo assim, como seria possível criar uma rede de experimentação sem interferir na rede de produção?

Outro fato é a complexidade de integração dos equipamentos de redes produzidos por diferentes fornecedores a fim de trabalharem em conjunto dentro de uma rede de computadores. Torna-se complexo integrar uma rede composta por equipamentos de um determinado fornecedor X com equipamentos de outro fornecedor Y, pois cada fornecedor programa e trata a solução de um determinado problema de acordo com a sua *API* proprietária. Por não termos acesso à forma como essas soluções são implementadas temos assim uma rede de computadores composta por várias "caixas pretas". Além de todos os problemas e dificuldades citados, ainda existe o desafio da gerência da rede a partir do ponto de vista do administrador da rede. Com inúmeros protocolos e soluções que atendem a finalidade diferentes, torna-se complexo o controle, a solução dos problemas e a visualização global da rede.

Através da tecnologia de redes definidas por *software* temos um novo conceito de arquitetura de redes, onde há uma visão e gerência global da rede além de outros benefícios como: a independência de fornecedores de equipamentos de redes (roteadores e *switches*), espaço para inovação e redução de complexidade através de automatização. Nesta tecnologia propõe-se a separação completa do plano de controle e do plano de dados, de maneira que o plano de controle passe a ser programável, possibilitando assim que o funcionamento da rede passe a ser definido a nível de aplicação. A principal ideia é permitir que os desenvolvedores de *softwares* possam contar com os recursos da rede da mesma forma como fazem com os recursos de armazenamento e computação. A fim de concretizar isso, é proposto o protocolo de código aberto OpenFlow (MCKEOWN, 2008), o qual já

foi adotado pela empresa Google (JAIN, 2013), para soluções de melhoria de engenharia de tráfego em seus próprios *backbones*. Assim, separando o plano de controle do plano de dados, a empresa tem a possibilidade de escolher o *hardware* específico baseado nas características que atendam às suas necessidades, enquanto é capaz de inovar no desenvolvimento das suas soluções. A partir do controle centralizado que as redes definidas por *software* fornecem, a Google foi capaz de desenvolver soluções mais específicas para seus problemas e que atendam às suas necessidades com a facilidade de serem customizadas. Sem contar a possibilidade de proporcionar flexibilidade à rede e um ambiente para inovação.

Tecnologias baseadas no protocolo OpenFlow permitem enfrentar problemas relacionados à dinâmica necessidade das aplicações atuais, adaptação das redes a cada mudança dos negócios e redução da complexidade no gerenciamento dessas aplicações. Assim sendo, os sistemas de gerenciamento de redes passam a ser adaptados a esta visão centralizada da rede, a qual esta tecnologia nos fornece, implicando diretamente na funcionalidade dos algoritmos para seleção de caminho, onde o processamento e o cálculo são executados a partir de um ponto lógico da rede, o qual possui uma visão global da topologia.

A rede definida por *software* leva a mudanças significativas no modo de criação e de operação das redes de computadores. As atuais redes de computadores apresentam diversas demandas que ainda precisam ser discutidas e tratadas. O conceito de redes definidas por *software* está atrelado à capacidade de inovação. Através desta tecnologia, os seus utilizadores podem personalizar e desenvolver novas soluções para as redes de acordo com as suas necessidades e demandas, eliminando ferramentas desnecessárias e criando redes totalmente virtuais e isoladas. SDN é um tecnologia ainda bastante recente, mas está crescendo em um ritmo muito rápido. Ainda assim, há importantes desafios na pesquisa a serem abordados, tratados e discutidos.

1.1 CONTEXTO E MOTIVAÇÃO

Através de décadas a Internet vem mudando a forma de como as pessoas se comunicam. A facilidade desta forma de comunicação deixa transparente aos seus usuários como a infraestrutura da Internet sofreu modificações ao longo do tempo. O surgimento de novos serviços que demandem mais qualidade emergiram, fazendo com que a rede tivesse que se adequar a essas mudanças rapidamente. Assim, as redes de computadores foram crescendo e se adaptando à grande quantidade de novas aplicações e protocolos.

A atual arquitetura das redes consiste em grande parte de conjuntos distintos de protocolos projetados para conectar os equipamentos de forma confiável, com topologias para satisfazer os negócios e necessidades técnicas emergentes ao longo dos anos. Os protocolos tendem a ser definidos de uma maneira isolada, cada um resolvendo um problema específico. Isso resultou em uma das principais limitações das redes de hoje: complexidade.

Os métodos de balanceamento de carga são em sua maioria complexos de serem implementados nas redes atuais, pois os algoritmos e soluções propostas são limitadas pelos fabricantes e disponíveis em *hardwares* proprietários. Analisando esta arquitetura, vemos que esses *hardwares* são fechados, com milhões de linhas de código fonte para a construção e administração do seu Sistema Operacional e das suas aplicações. Esses equipamentos requerem um desempenho computacional enorme, pois processam e encaminham informações ao mesmo tempo, consumindo bastante recursos de *hardware*. Esses fatores geram dificuldade em inovação e criam uma dependência de fornecedor. As arquiteturas de redes existentes não foram projetadas para atender os requisitos dos usuários atuais e por isso tornam a rede cada vez mais limitada. Surge então esta grande necessidade de gerenciar toda essa demanda e conteúdo de forma ágil e segura.

Os atuais equipamentos de redes possuem duas camadas distintas, como apresenta a Fig. 1.1. Existe uma camada de *software* (*software control*) responsável pelo gerenciamento dos protocolos e uma camada de *hardware* (*hardware datapath*) responsável por executar o encaminhamento dos pacotes. Apesar de serem camadas distintas, elas residem no mesmo equipamento, podendo ser separadas, pois executam funções totalmente diferentes. Geralmente esses equipamentos de redes são proprietários com *hardware*, sistema operacional e aplicações em um único equipamento.

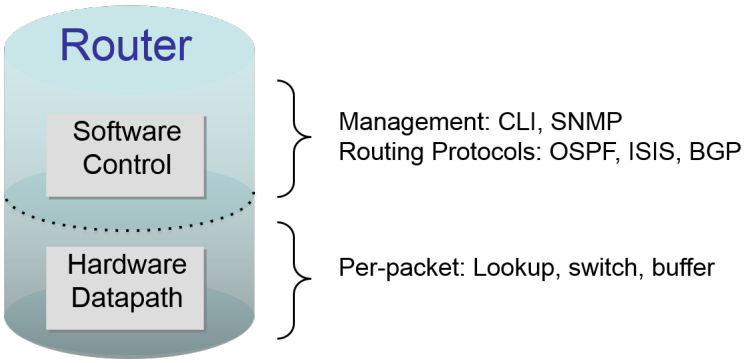


FIG. 1.1: Divisão em Camadas dos Equipamentos de Redes (MCKEOWN, 2010)

Para se implementar uma nova política nesta arquitetura de redes atuais, é necessário configurar milhares de dispositivos e mecanismos, algumas vezes de forma manual e de total dependência do administrador da rede, dificultando assim a sua gerência. Não há uma visão geral da topologia da rede onde seus recursos possam ser administrados de uma forma centralizada e unificada. Não há espaço para inovação, pois a rede fica limitada às soluções proprietárias e a protocolos fechados fornecidos pelos fabricantes dos dispositivos de redes. Com esses problemas, a rede acaba tornando-se complexa de se administrar, o que resulta em despesas na manutenção das suas operações.

Os problemas apresentados demonstram a necessidade da redução da complexidade das redes atuais, de forma que a administração possa ser realizada de forma mais centralizada, onde os recursos possam deixar de ser proprietários, trazendo assim mais espaço para inovação e melhorias nas redes atuais. Vimos que os protocolos e soluções existentes são em sua maioria complexos e limitados pelos seus fabricantes. Vimos que falta espaço para inovação e para reprodução real desses problemas a fim de se propor melhorias. As redes atuais tornaram-se "ossificadas" e muito limitadas para atenderem às demandas das aplicações emergentes. Com isso observamos o surgimento de novas tecnologias, com a proposta de uma outra abordagem do problema, onde assim poderemos obter uma visão completa da rede, podendo entender melhor os problemas a fim de propormos melhores soluções. Além disso será possível fomentar inovação aberta e competição, tornando assim a inovação da rede mais fácil e rápida. Toda essa ideia da tecnologia SDN, com a separação das camadas dos equipamentos de redes atuais, não surgiu por acaso, ela já vem florescendo ao longo dos anos, através de estudos sobre o quanto a rede tornou-se complexa. SDN já é realidade e foi adotada por uma das grandes empresas de tecnologia, a Google.

1.2 OBJETIVO DA DISSERTAÇÃO

A contribuição deste trabalho é avaliar as vantagens da utilização da tecnologia de redes definidas por *software* no balanceamento de carga, tendo em vista a visão centralizada da rede. Como o balanceamento de carga é um assunto bastante abordado em várias pesquisas, é interessante estudar o seu comportamento em uma estrutura como a SDN, pois a mesma possui uma arquitetura diferente das redes atuais de computadores. Além disso, será realizado um estudo dos diversos módulos e mecanismos que compõem uma rede de computadores baseada no protocolo OpenFlow. Assim será possível avaliar as van-

tagens oferecidas por essa tecnologia em comparação com os métodos de balanceamento de carga já existentes.

O objetivo desse trabalho é a pesquisa e também a construção de um aplicação de balanceamento de fluxos, chamada LBX, em uma rede de computadores com suporte a múltiplos caminhos. Será proposto um algoritmo de seleção de caminho de custo mínimo, baseado no algoritmo de Dijkstra. O objetivo do algortimo de seleção de caminho é balancear os fluxos da rede baseando-se sempre no caminho de custo mínimo. Esses fluxos serão identificados e separados a nível de aplicação, através do valor do campo *ToS* obtido através do cabeçalho OpenFlow do fluxo analisado pelo controlador da rede. Com isso, poderemos balancear os fluxos a partir das aplicações identificadas na rede. Todos os caminhos de custo mínimo, calculados pelo algoritmo, serão armazenados em um banco de dados da própria aplicação, para que possam ser utilizados a fim de recalcular um outro caminho disjunto de custo mínimo na rede. Com isso, utilizaremos caminhos disjuntos sem interseção de enlaces para o balanceamento dos fluxos. Assim poderemos utilizar todo o recurso da rede, fazendo com que os nós não fiquem sobrecarregados ou sub-utilizados. A partir disto, os fluxos serão balanceados pelo enlaces de maneira justa, respeitando a necessidade das aplicações definidas pelo administrador da rede.

Por meio de experimentos avaliaremos quais serão os benefícios obtidos a partir de uma visão centralizada e global da rede, além do comportamento do algoritmo proposto. Serão realizados experimentos isolados em dois cenários distintos (C_1 e C_2), onde compararemos os benefícios do balanceamento de carga, a visão centralizada da rede, o balanceador de fluxos LBX proposto e a solução ECMP (*Equal Cost Multipath Routing*) implementada pela empresa Cisco, melhor detalhada no Capítulo 2 desse trabalho. Com isso, poderemos identificar melhor o problema do balanceamento de carga e apresentar os benefícios de uma visão centralizada para este problema. Avaliaremos a possibilidade de inovação no assunto com a finalidade de modificar, tratar e avaliar melhor os problemas apresentados.

Sendo redes definidas por *software* uma tecnologia emergente, de código aberto e de livre acesso para a criação de laboratórios, poderemos explorar mais a ferramenta a fim de podermos melhor avaliar as vantagens e desvantagens desta tecnologia. Com isso, apresentaremos os resultados obtidos através de um ambiente utilizando o controlador Floodlight, implementado pela empresa Big Switch Networks e pela maior comunidade de desenvolvedores do mundo para controladores SDN, e através do emulador de redes Mininet, bem referenciado em publicações sobre o tema de SDN, capaz de construir uma

rede virtualizada.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

O presente trabalho está organizado da seguinte forma:

- Capítulo 2 - *Roteamento de Múltiplos Caminhos*, é apresentado o conceito de roteamento de múltiplos caminhos com seus benefícios e suas desvantagens e os modelos de encaminhamento que realizam o balancemanento de carga.
- Capítulo 3 - *Redes Definidas por Software*, são apresentados os conceitos teóricos sobre SDN, sua origem e história, sua arquitetura, o protocolo OpenFlow e seus componentes: *switches* e controladores. Destacamos o controlador Floodlight e o emulador de rede Mininet, que serão utilizados neste trabalho.
- Capítulo 4 - *Arquitetura Proposta*, é apresentada a arquitetura e o algoritmo propostos neste trabalho.
- Capítulo 5 - *Implementação*, detalha a implementação, bem como as ferramentas utilizadas para viabilizar o desenvolvimento da solução proposta.
- Capítulo 6 - *Resultados*, demonstra os resultados obtidos com a utilização da proposta. Para isso são apresentados os testes realizados e as topologias avaliadas nos experimentos.
- Capítulo 7 - *Considerações Finais*, traz as conclusões e as perspectivas para os trabalhos futuros.
- Capítulo 8 - *Referências Bibliográficas*, lista as referências utilizadas como base para o presente trabalho.

2 ROTEAMENTO DE MÚLTIPLOS CAMINHOS

O Roteamento de Múltiplos Caminhos (acrônimo em inglês *Multipath Routing*) é uma técnica de roteamento que utiliza múltiplos caminhos alternativos para o tráfego de dados dentro de uma rede de computadores (PRABHAVAT, 2011). Esta técnica pode produzir benefícios, tais como:

- Balanceamento de Carga - através de soluções de balanceamento é possível distribuir melhor a carga entre todos os nós e enlaces da topologia da rede, de maneira mais justa impedindo que um determinado nó sofra sobrecarga em sua camada de dados, responsável por realizar o encaminhamento dos pacotes. A necessidade de uso de balanceamento de carga nas redes de computadores atrai muita atenção de pesquisas ao redor do mundo, pois realizá-lo pode-se tornar uma tarefa complexa.

- Customização de Determinadas Aplicações - na redes atuais, diferentes aplicações possuem necessidades diferentes. Com o surgimento da Qualidade do Serviço (conhecida como QoS, acrônimo do inglês, *Quality of Service*) é possível oferecer maior garantia e segurança para as aplicações, uma vez que o tráfego de determinadas aplicações passam a ser priorizados, a fim de serem trafegados com mais qualidade dentro da rede. Com o uso de QoS aliado à técnica de roteamento de múltiplos caminhos é possível que as aplicações sejam priorizadas dentro da rede e que seus tráfegos sejam trafegados por vários caminhos simultaneamente. Com isso é possível garantir a customização do trágefo de determinadas aplicações dentro da rede.

- Tolerância a Falhas - em situações onde ocorram falhas de comunicação entre os nós da topologia da rede, é possível, com o uso de roteamento de múltiplos caminhos, fazer com que tráfego seja redirecionado para um outro nó disponível através de caminhos alternativos, evitando assim perda dos pacotes trafegados pela rede.

A utilização de técnicas já existentes para o roteamento do tráfego das redes atuais aliado ao roteamento de múltiplos caminhos podem agregar benefícios para a engenharia de tráfego da rede. Uma das funções da engenharia de tráfego é controlar os fluxos em uma infraestrutura de transporte, de modo a atender critérios definidos pela operação

da rede e pelos requisitos dos fluxos. Esses fatores combinados melhoram a condição de performance e de eficiência da rede, pois influenciam no balanceamento da carga e tornam a rede mais tolerante a falhas.

A Fig. 2.1 apresenta o modelo de roteamento de múltiplos caminhos de forma mais generalizada, onde a origem ou um *gateway* da rede distribui o tráfego através de vários caminhos distintos até o seu respectivo destino. Desta forma o tráfego entre a origem e destino pode ser distribuído ao longo de vários caminhos realizando assim o balanceamento da carga da rede.

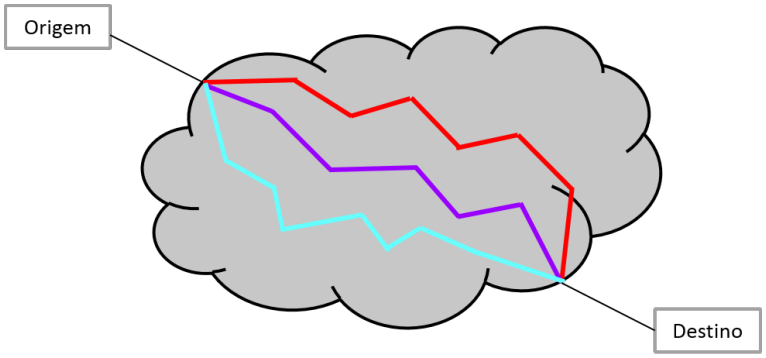


FIG. 2.1: Modelo do Roteamento de Múltiplos Caminhos

Enquanto o roteamento por múltiplos caminhos facilita a distribuição de tráfego entre uma origem e um destino, a forma como esses pacotes são divididos é uma questão importante. Segundo (PRABHAVAT, 2011) o encaminhamento de tráfego através de múltiplos caminhos é realizado basicamente através de dois componentes principais: a divisão de tráfego (*traffic splitting*) e a seleção do caminho (*path selection*). A Fig. 2.2 ilustra os componentes desse encaminhamento. O componente da divisão de tráfego divide o tráfego de dados em unidades menores e distribui essas unidades independentemente por vários caminhos baseado em um componente da seleção do caminho. Caso o processador de encaminhamento esteja ocupado, cada unidade de tráfego é enfileirada em uma saída por um componente de seleção de caminho. O componente da seleção do caminho é responsável por escolher um caminho para cada pacote.

Em nível de divisão de tráfego baseado em pacote, o tráfego é dividido em menor escala possível, isto é, em um único pacote. Com isso a seleção do caminho é decidida individualmente por cada pacote. O modelo de distribuição de carga desse tipo de separação de tráfego é referenciado como um modelo de distribuição de carga baseada em pacotes.

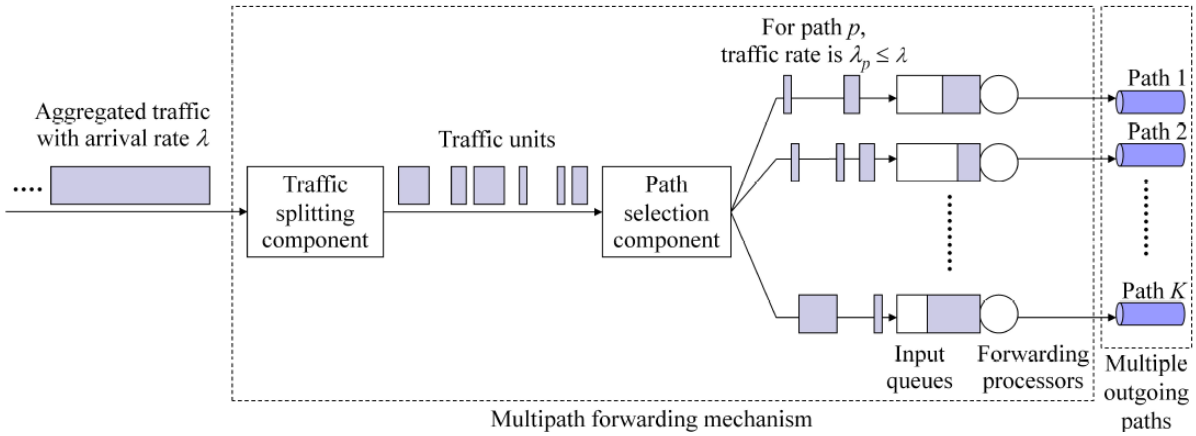


FIG. 2.2: Componentes do mecanismo de encaminhamento através de múltiplos caminhos (PRABHAVAT, 2011)

Em nível de divisão de tráfego baseado em fluxo, informações sobre o tráfego da rede, são levados em consideração na divisão do tráfego. Todos os pacotes com o mesmo destino são agrupados, cada grupo é definido como uma unidade do fluxo com um identificador único. A divisão de tráfego a este nível pode garantir a ordenação dos pacotes idêntica desde a seleção do caminho para todos os pacotes do mesmo fluxo.

A seleção do caminho é responsável por escolher o caminho para onde cada pacote será trafegado e ela é feita independentemente para cada fluxo. O modelo de distribuição de carga para este tipo de separação de tráfego é referenciado como um modelo de distribuição de carga baseada em fluxos. A maioria dos modelos de seleção de caminho podem ser classificados em quatro tipos, como mostra a Fig. 2.3.

FIG. 2.3: Taxonomia da seleção do caminho. Adaptado de (PRABHAVAT, 2011)

O seletor aleatório é um esquema de seleção de caminho onde as unidades de tráfegos sucessivamente são enviadas através de todos os caminhos paralelos de maneira aleatória.

No seletor baseado em informação do pacote (*PacketInfo*), o identificador do pacote, obtido através das informações contidas no cabeçalho do pacote, é de importante decisão para a escolha da seleção do caminho. Tipicamente a escolha de um caminho é decidida baseada em uma função do identificador do pacote. No seletor baseado nas condições de tráfego, informações sobre o tráfego são levadas em consideração na escolha do componente de seleção do caminho. Estas incluem: quantidade de tráfego, taxa do tráfego e número de tráfegos ativos. E por fim, no seletor baseado nas condições da rede, fatores como atraso (*delay*), perda de pacotes (*packet-loss*) e tamanho do *buffer* são levados em consideração para determinar a seleção de caminho segundo o objetivo do balanceamento de carga.

Infelizmente, grande parte desta diversidade de caminhos alternativos é pouco explorada. O desafio da falta de escalabilidade do roteamento de múltiplos caminhos é uma dessas razões. Esta técnica de roteamento produz uma sobrecarga extra tanto no plano de controle quanto no plano de dados dos nós. No plano de controle, os nós calculam as tabelas de encaminhamento que o plano de dados usa para direcionar os pacotes de entrada e de saída em um determinado nó da rede.

Atualmente, na Internet, existem dois principais protocolos de roteamento: o *Border Gateway Protocol* (BGP) (REKHTER, 2006) e o *Interior Gateway Protocol* (IGP) (GROSS, 1992). O BGP É um protocolo de roteamento dinâmico, de inter-domínio baseado em algoritmo vetor distância, onde as decisões de roteamento são realizadas com base em políticas locais. Já o IGP é um protocolo de roteamento interno, usado por *gateways* de um mesmo sistema autônomo. Os protocolos RIP (*Routing Information Protocol*) (MALKIN, 1998), OSPF (*Open Shortest Path First*) (MOY, 1998) e IGRP (*Interior Gateway Routing Protocol*) (HEDRICK, 1991) são exemplos de IGP. O EIGRP (*Enhanced Interior Gateway Routing Protocol*) é uma evolução do IGRP.

O RIP é um protocolo de roteamento, baseado no algoritmo Bellman-Ford, que utiliza vetores de distância no intuito de comparar matematicamente rotas para identificar o melhor trajeto a todo o endereço de destino dado. Este algoritmo foi usado para os cálculos de encaminhamento em redes de computadores, desde os primeiros dias da ARPANET (SIMON, 1997). O RIP utiliza um único roteamento métrico (contagem de saltos) para medir a distância entre dois nós e possui uma limitação de 15 saltos. Atualmente encontra-se na sua segunda versão RIP 2.

O OSPF é um protocolo que executa um algoritmo de estado de enlace, onde as decisões de roteamento são tomadas a partir do caminho mais curto dentro da rede baseado

configurações de pesos nos enlaces entre os nós. Atualmente o OSPF é um dos protocolos de roteamento mais empregados em redes de computadores, sendo suportado pela maioria dos roteadores. Por ser um protocolo versátil, o OSPF pode ser empregado tanto a redes de pequeno ou grande porte. Assim como o RIP, OSPF também encontra-se em sua segunda versão.

O IGRP é um protocolo proprietário desenvolvido na década de 80, pela empresa Cisco, com o objetivo de fornecer um protocolo robusto para ser distribuído dentro de um sistema autônomo. Assim como o RIP, o IGRP utiliza vetores distância na comparação de rotas além de identificar o melhor caminho para cada caminho.O IGRP resolveu grande parte dos problemas associados ao uso do RIP para roteamento interno. O algoritmo utilizado pelo IGRP determina o melhor caminho entre dois pontos dentro de uma rede, examinando a largura de banda e o atraso de envio de pacotes entre os nós. O IGRP converge mais rapidamente que o RIP, evitando loops de roteamento e não tem a limitação de saltos entre nós.

O EIGRP representa uma evolução do seu antecessor IGRP. É um protocolo avançado de roteamento por vetor distância proprietário da empresa Cisco. O protocolo EIGRP proporciona compatibilidade e interoperação direta com os roteadores IGRP. Um mecanismo de redistribuição automática permite que os roteadores IGRP sejam incorporados para EIGRP e vice-versa, possibilitando assim, adicionar gradualmente o EIGRP a uma rede IGRP existente.

Tanto as políticas locais do BGP quanto os pesos dos enlaces do OSPF são configurados manualmente por administradores da rede com a finalidade de atingir estratégias e objetivos de negócios. No IGP o roteamento é baseado em uma única métrica, por exemplo, no OSPF o peso dos enlaces que cada nó possui ou os pesos dos enlaces associados. Por outro lado, mesmo que cada nó possa ter uma visão completa da rede, a diversidade de caminhos existentes ainda é sub-aproveitada. Mesmo quando os caminhos alternativos foram calculados, os pacotes de dados para um determinado destino são frequentemente enviados para um único caminho.

2.1 MODELOS DE CLASSIFICAÇÃO DA DISTRIBUIÇÃO DE CARGA

Vários modelos de encaminhamento de múltiplos caminhos realizam a distribuição de carga nas mais diferentes maneiras. Cada modelo apresenta diferentes vantagens e deficiências, devido à diferença em seus componentes internos. De acordo com (PRABHAVAT, 2011) os

modelos de distribuição de carga existentes podem ser classificados como não-adaptativos e adaptativos, como apresentados na Fig. 2.4.

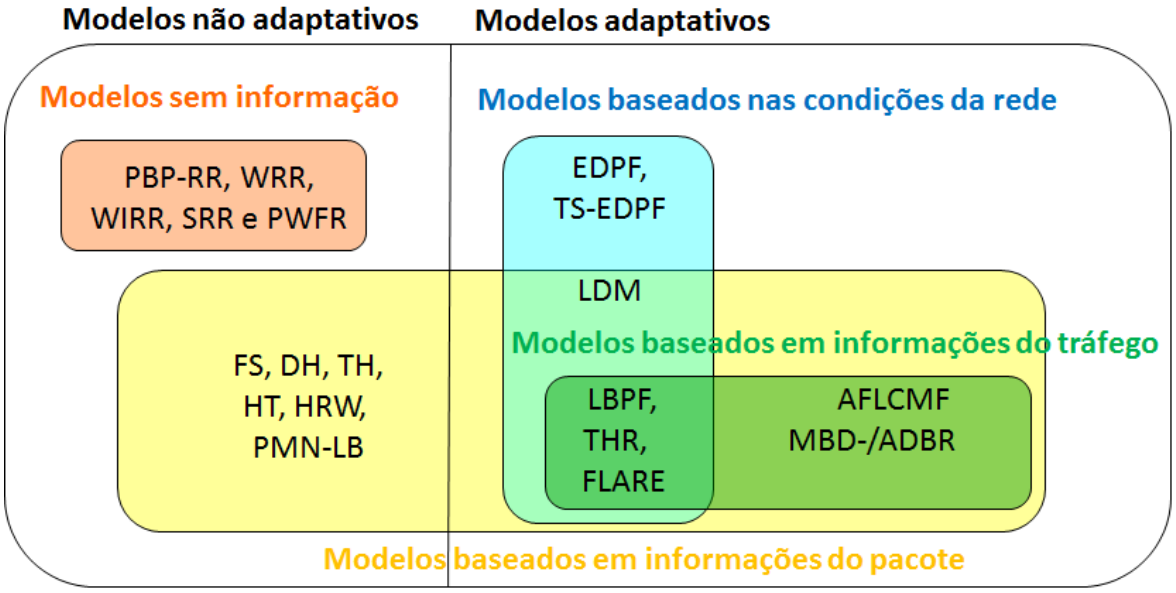


FIG. 2.4: Modelos de classificação da distribuição de carga. Adaptado de (PRABHAVAT, 2011)

2.1.1 CLASSIFICAÇÃO DOS MODELOS NÃO-ADAPTATIVOS

Os modelos pertencentes à classe Modelos sem informação tomam uma decisão de distribuição de tráfego sem levar em consideração nenhuma informação externa e os pertencentes à classe Baseados em Informações do Pacote necessitam de informações obtidas a partir do cabeçalho do pacote.

2.1.1.1 MODELOS SEM INFORMAÇÃO

Esses modelos não coletam nenhuma informação sobre o tráfego ou sobre as condições da rede.

Packet-By-Packet Round-Robin (PBP-RR)

NO PBP-RR (VILLAMIZAR, 1999), a divisão do tráfego é definida de maneira aleatória, onde não existe um caminho ocioso enquanto um pacote está esperando para ser

enviado. Devido a sua incapacidade de controlar a quantidade de carga compartilhada, o PBP-RR não é capaz de equilibrar a carga entre vários caminhos heterogêneos. Se o parâmetro de cada caminho for diferente, o PBP-RR pode causar problemas como o excesso de utilização de um caminho com baixa capacidade e subutilização de um caminho com alta capacidade.

Weighted Round Robin (WRR)

No modelo WRR (PAREKH, 1993), a cada caminho é atribuído um valor que significa, em relação aos outros caminhos no conjunto de vários caminhos, quanta carga de tráfego deve ser atribuída neste caminho de conexão. Este valor. o qual podemos chamar de peso, determina quantos pacotes serão enviados pelo caminho em comparação aos outros caminhos, ou seja, o número de pacotes designados para os caminhos estão limitados pelos pesos. No WRR, o desiquilíbrio da carga pode ocorrer devido à variação no tamanho dos pacotes. O WRR já foi incorporado a vários protocolos como o EIGRP.

Weighted Interleaved Round Robin (WIRR)

O WIRR (LENGYEL, 2005) e (LENGYEL, 2004) possui características semelhantes ao WRR exceto que um pacote sucessivo será enviado para o próximo caminho de maneira aleatória. Somente os caminhos que possuem um menor número de pacotes enviados que o número desejado permanecerão em um pool (de caminhos a serem selecionados) para a próxima iteração. Ao contrário do WRR, o WIRR impede a utilização contínua de um caminho específico. Assim como o WRR, o WIRR também possui problemas para manter ordenados os pacotes por fluxos.

Surplus Round Robin (SRR)

O SRR (ADISESHU, 1996) é baseado em uma modificação do WRR onde o SRR tem um desempenho melhor no balanceamento de carga, pois ele usa um contador de byte e não é afetado pela variação do tamanho do pacote. Cada caminho é associado a um contador de *déficit* de serviço medido em bytes, proporcional à largura de banda do percurso. Cada vez que um caminho é selecionado para o envio de um pacote, seu contador *déficit* é decrementado pelo tamanho do pacote. Contanto que o contador de *déficit* seja positivo, o resultado da seleção permanecerá inalterado. Caso contrário, o próximo caminho com *déficit* positivo será selecionado em uma maneira aleatória.

Packet-By-Packet Weighted Fair Routing (PWFR)

O PWFR (LEUNG, 2006) foi projetado para realizar eficazmente o balanceamento da carga. Nele cada caminho tem um determinado peso de encaminhamento indicando a quantidade de carga a ser balanceada, onde o termo "carga" é o número de bytes de um pacote. Para cada chegada de pacote, o contador de *déficit* de cada caminho é aumentado por uma fração do tamanho do pacote para aquele caminho. Este modelo possui um nível computacional de complexidade *O(K)* onde o tempo de processamento da seleção de caminho por cada pacote aumenta quando o número de caminhos aumenta.

2.1.1.2 MODELOS BASEADOS EM INFORMAÇÕES DO PACOTE (MODELOS NÃO ADAPTATIVOS)

A reordenação dos pacotes é o principal problema dos modelos sem informação. Selecionar o mesmo caminho para todos os pacotes com o mesmo endereço de destino pode solucionar esse problema. Para realizá-lo, a informação do pacote é necessária para a escolha do caminho (THALER, 2000), (CAO, 2000) e (HOPPS, 2000).

Fast Switching (FS)

O FS (ZININ, 2002) é um modelo baseado em fluxos que é uma tecnologia proprietária da *Cisco*. Em um mesmo fluxo, os pacotes são enviados através do mesmo caminho. Quando um novo fluxo surge, os pacotes pertencentes ao novo fluxo são enviados através do próximo caminho paralelo em uma maneira aleatória e o mapeamento do caminho deste novo fluxo é armazenado na memória cache. O FS realiza o balanceamento do número de fluxos entre os caminhos, porém ele não pode lidar com assimetria da distribuição do tamanho do fluxo o que pode causar desiquilíbrio de carga. Além disso, o FS requer memória para armazenar o estado do fluxo. Caso um novo fluxo surge enquanto não houver espaço disponível na memória *cache*, a entrada do mapeamento de fluxo mais antigo é removido. Como consequência, o caminho do fluxo mais antigo pode mudar resultando na reordenação dos pacotes. Por isso é essencial que o espaço da memória em cache seja de tamanho suficiente para manter os registros de mapeamento dos fluxos.

Direct Hashing (DH)

O DH é um modelo baseado em fluxo convencional que é amplamente implantado em protocolos de roteamento de múltiplos caminhos (VILLAMIZAR, 1999), (THALER,

2000) e (HOPPS, 2000). Ele realiza o balanceamento da carga baseado na realização de *hash* para as rotas ECMP (*Equal Cost Multipath Routing*). Para obter o caminho de saída, ele executa um algoritmo de *hash* módulo-K que leva em consideração o identificador do pacote X, obtido através das informações do cabeçalho do pacote. Assim, aplica uma função *hash h(X)* e obtém o módulo do número dos múltiplos caminhos *mod(h(X))*. O algoritmo é simples com complexidade computacional de *O(1)*, no entanto, o desempenho do balanceamento da carga depende da distribuição de valores *hash*. Pode ocorrer de todos os fluxos possuírem o mesmo valor *hash* e assim todos os pacotes serão enviados através de um único caminho, resultando assim no desiquilíbrio de carga. O DH não lida com a variação da distribuição do tamanho do fluxo. O DH pode alcançar uma ótima performance de balanceamento de carga quando os resultados da função *hash* são uniformemente distribuídos.

Table-Based Hashing (TH)

O TH (VILLAMIZAR, 1999) é um esquema baseado em *hash* usado no roteamento ECMP. Cada fluxo é assinalado a um caminho, de acordo com uma tabela de mapeamento. Cada fluxo corresponde a um *bin*. Este *bin* implica em fluxos que possuem o mesmo valor de identificador de fluxo e *hash*. O TH permite distribuir tráfego simplesmente modificando o valor do *bin* para os caminhos. Este esquema possui uma complexidade de *O(1)* e possui problemas ao lidar com variações de tamanho de fluxos.

Hash Threshold (HT)

O HT (VILLAMIZAR, 1999) é um esquema de balanceamento de carga incorporado no roteamento ECMP e possui características semelhantes ao modelo TH, exceto a tabela de mapeamento. Assim como o TH, o HT também realiza o *hash* dos fluxos porém ele particiona o *hash* em regiões. Cada região é um conjunto de identificações do fluxo que será encaminhado através de um caminho. A fim de alcançar a partilha equitativa da carga, o *hash* é igualmente dividido por todas as regiões K de mesmo tamanho. A fim de realizar um balanceamento de carga igual, o resultado da função *hash* é igualmente dividido para todas as K regiões do mesmo tamanho. Um caminho deve ser supostamente escolhido por um pacote entrante podendo ser determinado por achar a região que contém o resultado *hash* do pacote enviado anteriormente.

Highest Random Weight (HRW)

O HRW (THALER, 1998) é um esquema de balanceamento de carga usado em caches WWW e no roteamento ECMP. No HRW um caminho é selecionado com base no seu peso aleatório, calculado com base no identificador do pacote (X), e no identificador do caminho (r_i). Sempre o caminho com o maior peso aleatório é selecionado. Em comparação com o DH e o TH, o HRW possui uma maior complexidade computacional *O(K)*.

Primary Number Modulo-N Load Balance (PMN-LB)

O PMN-LB (KIM, 2006) e (KIM, 2006) utiliza dois algoritmos para a seleção do caminho: algoritmo primário e secundário. O algoritmo primário realiza o *hash* comum módulo-N, semelhante ao DH, para todos os fluxos. O algoritmo primário é executado para a seleção do caminho. No entanto, quando o número de caminhos disponíveis muda, é possível que, sem atualizar o divisor N, o algoritmo de *hash* módulo-N não possa selecionar os caminhos disponíveis para alguns fluxos, pois os caminhos escolhidos podem não estar disponíveis no momento. Se isso acontecer, o algoritmo secundário será executado para assegurar a seleção de um caminho disponível para os fluxos. Em comparação ao HRW, o PMN-LB oferece um menor desempenho de pesquisa, *O(1)*.

2.2 CLASSIFICAÇÃO DOS MODELOS ADAPTATIVOS

O modelos analisados até agora não levam em consideração as condições dinâmicas dos tráfegos e da rede. O modelos adaptativos podem ser usados para resolver estes problemas. Classificamos os modelos adaptativos em duas classes de acordo com os respectivos tipos de condições.

2.2.1 MODELOS BASEADOS NAS INFORMAÇÕES DO TRÁFEGO

Os modelos de balanceamento de carga adaptativos pertencentes a esta classe podem adaptar-se às condições do tráfego, incluindo a quantidade de carga de um fluxo (em pacotes ou bytes) como bem as características do tráfego.

Adaptive Flow-Level Load Control Scheme for Multipath Forwarding (AFLCMF)

AFLCMF (LEE, 2001) permite distribuir o tráfego entre vários caminhos numa pro-

porção pré-definida. Essa razão e uma velocidade de chegada do pacote são levados em conta na determinação de um limite de taxa utilizada para a classificação do fluxo. Cada fluxo, o qual é classificado de acordo com base na taxa de chegada de pacotes é enviado através de uma via selecionada correspondente à sua classe. No entanto, por se ajustar à condição do tráfego, vários fluxos podem experimentar mudanças de classes resultando assim na comutação do percurso. Os tempos de processamento de classificação do fluxo e seleção do caminho possuem complexidade *O(K)*.

Progressive Multiple Bin Disconnection with Absolute Difference Bin Reconnection (MBD-/ADBR)

O MBD-/ADBR (MARTIN, 2006) é uma versão do modelo TH porém a tabela de mapeamento dos caminhos dos fluxos pode ser alterada dinamicamente. O número de pacotes em cada fluxo é levado em consideração na determinação do tamanho do fluxo e no estado do caminho. A carga real, que é o número total de pacotes transmitidos através de cada percurso, é utilizada quando o caminho está sendo utilizado ou sub-utilizado. Cada fase de controle consiste em duas etapas. No primeiro passo, o mais pequeno fluxo associado ao caminho mais sobrecarregado é removido, assim torna-se um fluxo sub-utilizado. O segundo passo é atribuir o maior fluxo ao caminho mais subutilizado repetidamente até que todos os caminhos não estejam mais sobrecarregados. Redistribuir a carga excessiva através dos caminhos sub-utilizados com frequência pode melhorar a eficiência do balanceamento de carga mas causar risco de reordenação dos pacotes.

2.2.2 MODELOS BASEADOS NAS INFORMAÇÕES DA REDE (MODELOS ADAPTATIVOS)

Para os modelos dessa classe, as condições da rede como utilização e tempo de entrega são levados em consideração para seleção do caminho.

Earliest Delivery Path First (EDPF)

O EDPF (CHEBROLU, 2006) foi proposto para o balanceamento de carga em redes *wireless* e para ser implementado em dispositivos equipados com múltiplas interfaces. A interface correspondente é ativada quando um caminho é selecionado. O objetivo do modelo EDPF é assegurar que os pacotes cheguem ao seu destino com uma certa duração através de agendamento de pacotes baseados no tempo estimado de entrega. O EDPF

considera as características do caminho, como atraso e largura de banda entre origem e destino. O balanceamento de carga do EDPF é limitado pelo tamanho máximo do pacote. O EDPF possui uma complexidade computacional de *O(K)*.

Time-Slotted Earliest Delivery Path First (TS-EDPF)

O TS-EDPF (FERNANDEZ, 2009) é uma versão melhorada do EDPF e tem como objetivo fornecer capacidade de gerenciamento para um servidor QoS (*Quality of Service*) para alocação de banda para cada estação móvel a fim de reduzir o tempo de espera na fila de pacotes. O TS-EDPF modifica o algoritmo de escalonamento para decidir a seleção do caminho. A sua escalabilidade é semelhante à do EDPF e ele pode garantir qualidade de serviço e redução de entrega de pacotes.

Load Distribution over Multipath (LDM)

O LDM (SONG, 2003) é um modelo de distribuição de carga baseado no conceito de Engenharia de Tráfego (AWDUCHE, 2002), projetado para redes MPLS (*Multiprotocol Label Switching*) (ROSEN, 2001). Para cada fluxo entrante, a utilização do caminho até o momento e o salto de contagem do caminho é usado para determinar a probabilidade da seleção do caminho. O LDM seleciona randomicamente um caminho. No entanto, o LDM não realiza a divisão do tráfego, com isso o balanceamento de carga pode ser degradado pela variação na distribuição do tamanho do fluxo. LDM possui uma complexidade computacional de *O(K)*.

2.2.3 MODELOS ADAPTATIVOS BASEADOS NAS INFORMAÇÕES DE TRÁFEGO E CONDIÇÕES DA REDE

Para os modelos dessa classe, ambas as condições de tráfego e da rede são levados em consideração para a seleção do caminho a fim de melhorar a distribuição da carga, como o balanceamento de carga (SHI, 2005) e (CHIM, 2005) e a preservação da ordem dos pacotes (KANDULA, 2007).

Load Balancing for Parallel Forwarding (LBPF)

O LBPF (SHI, 2005) é um esquema de balanceamento de carga adaptativo que visa

lidar com o desiquilíbrio da carga devido a distribuições de tamanho de fluxo altamente distorcidas. O LBPF seleciona o caminho para um fluxo de acordo com o resultado do *hash*, similar aos modelos convencionais baseados no *hash*, porém ele leva em consideração a taxa de tráfego de cada fluxo. O fluxos com alta taxa são classificados em um grupo de fluxos agressivos, quanto o sistema está sob uma condição específica (por exemplo, o sistema está desequilibrado). O algoritmo de adaptação é ativado, em tal condição, e cada pacote analisado possui uma marca onde pode-se identificar se o mesmo pertence a um fluxo agressivo ou não. Caso pertença a um fluxo agressivo, é ajustado para ser transmitido através do caminho mais curto. O LBPF pode lidar com a assimetria da distribuição de tamanho de fluxo e melhorar o desempenho do balanceamento de carga, no entanto, não pode lidar com o desequilíbrio da carga resultante dos fluxos que não são classificados como agressivos.

Table-Based Hashing with Reassignment (THR)

O TH (CHIM, 2005) é semelhante ao TH porém o mapeamento do caminho dos fluxos pode variar dinamicamente. Para isso um contador e temporizador são utilizados para registrar os números de pacotes e o tempo entre a chegada de cada pacote. A carga real, que é o número total de pacotes transmitidos através de cada caminho, é usado para determinar se o caminho é subutilizado ou não. O THR tem um parâmetro chave pré-determinado B, o qual determina a prioridade entre a melhoria do desequilíbrio da carga e prevenindo a reordenação dos pacotes. Este método tem dificuldade de preservar a ordem dos pacotes. O THR possui complexidade computacional de $O(K)$.

Flowlet Aware Routing Engine (FLARE)

O FLARE (KANDULA, 2007) foi proposto para alcançar o balanceamento de carga enquanto realiza a prevenção da reordenação dos pacotes para a distribuição de carga entre múltiplos caminhos. No FLARE um fluxo é dividido em vários sub fluxos, cada um é referido como um *flowlet*. Um parâmetro chave pré-determinado é um limite de tempo entre as chegadas dos pacotes. Um pacote entrante com uma duração inferior ao ultimo pacote trafegado é parte de um *flowlet* existente e por isso será enviado pelo mesmo caminho do último pacote. A divisão condicional dos fluxos é uma propriedade fundamental para o FLARE e sua complexidade computacional é de $O(K)$.

Como visto, todos os modelos apresentados possuem vantagens e desvantagens para a realização do balanceamento da carga em redes de computadores com suporte a múltiplos caminhos. Observamos que alguns modelos não levam em consideração nenhuma informação da rede e isto pode ser muito prejudicial para a seleção do caminho e para o tráfego das informações na rede. Desta maneira o caminho pode ser simplesmente calculado de forma aleatória, como nos modelos PBP-RR, WRR e IRR, havendo assim uma sobrecarga ou subutilização de alguns nós na topologia da rede, levando ao desequilíbrio da distribuição da carga. Os modelos que se baseiam nas informações obtidas através dos pacotes como o DH, HT, TH e o HRW realizam, em sua maioria, uma função *hash* com as informações obtidas nos pacotes. Esta função *hash* divide os pacotes em unidades menores e enviam essas unidades longo dos múltiplos caminhos. A realização dessa função *hash* pode influenciar em problemas de performance dentro da rede, pois desta maneira, enviando milhares de unidades menores de pacotes torna-se necessário algum método de reordenação dessas unidades na chegada ao seu destino. Este fator pode comprometer diretamente a eficiência do balanceamento da carga.

Já os modelos baseados nas informações sobre o tráfego da rede, são modelos que se adaptam as condições de tráfego como o AFLCMF e o MBD-/ADBR, porém também existe a necessidade da reordenação dos pacotes no seu destino. Apesar de serem adaptativos, as condições do tráfego desses métodos possuem um controle melhor sobre a administração dos caminhos já utilizados, em comparação aos métodos baseados somente na informações do pacote. Os modelos desta classe também precisam se preocupar com a reordenação dos pacotes enviados ao longos dos múltiplos caminhos. Os modelos baseados nas informações da rede também são adaptativos e as informações como a utilização da rede e tempo de entrega dos pacotes são levadas em consideração para a seleção do caminho como nos métodos EDPF, TS-EDPF e o LDM. Porém, quando a utilização da rede é alta, os algoritmos não permitem a mudança na seleção do caminho, como ocorre nos outros modelos que permitem que o fluxo seja re-roteado por outros caminhos.

Por fim, os modelos baseados nas condições do tráfego também apresentam desvantagens como o LBPF, THR e o FLARE, pois também necessitam da reordenação dos pacotes na chegada ao seu destino. A maioria dos modelos apresentados possuem uma visão parcial da rede, não encontramos nenhum modelo que obtenha uma visão global a fim de usá-la para a seleção do caminho dentro de uma rede de computadores com suporte a múltiplos caminhos.

Como visto existem algumas preocupações ao se escolher o método para seleção do caminho dentro de uma rede com suporte a múltiplos caminhos. Alguns fatores importantes já apresentados ainda precisam ser discutidos em questão para que os modelos possam levar em consideração: o desempenho, interrupção, a mudança de cada caminho que um fluxo de dados usa e a necessidade de reordenação dos pacotes na chegada a seu destino. Além disso, a necessidade de uma visão global da rede mostra-se importante para a seleção do caminho. Com isso, dois novos fatores são levados em consideração para a seleção de caminho dentro de uma rede com suporte a múltiplos caminhos.

- *Visão da rede* - nos modelos apresentados para a seleção do caminho, observamos que alguns são baseados em informações da rede, porém nenhum obtêm uma visão global da topologia a fim de conhecer todos os nós e enlaces ociosos ou sobrecarregados. A seleção do caminho é realizada somente e diretamente pelo nó, pois o modelo em questão é implementado no nó e o mesmo passa a ter o papel decisivo para a seleção do caminho. É válido ressaltar a importância de se utilizar a visão global da rede como parâmetro para seleção de caminho, com a finalidade de entender a topologia da rede e melhor distribuir o tráfego dos fluxos de maneira mais justa. Como este fator não é levado em consideração pelos modelos já existentes é importante considerar que isto poderia ajudar na seleção do caminho.

- *Visão do nó* - como já comentado, a seleção do caminho dos modelos apresentados é realizada diretamente em cada nó individualmente, fazendo com que cada nó sofra uma sobrecarga tanto no seu plano de controle como no plano de encaminhamento. Cada nó é responsável pela seleção do caminho e envio dos fluxos sem a influência da visão de rede neste processo. As informações da rede são compartilhadas pelos recursos, os quais atualizam suas próprias bases de dados periodicamente, o que poderia ocasionar em uma escolha baseada em informações não atualizadas sobre a rede.

A maioria dos algoritmos de seleção de caminho em redes com múltiplos caminhos não levam em consideração uma visão centralizada da rede e sim a visão parcial do nó, onde o algoritmo está sendo executado. Isso faz com que cada nó realize a sua própria seleção de caminho e envio de fluxos dentro da rede. Além disso, cada nó sofre um alto nível de processamento para poder computar seus algoritmos e definir a seleção do caminho dentro da rede. O desempenho dos modelos de distribuição de carga dependem em grande parte

do recurso dos esquemas na sua divisão de tráfego e caminho de seleção. As soluções desses algoritmos são complexas e limitadas para se implementar e administrar, cada fornecedor tem a sua própria solução para tratar de problemas específicos, ação encontrando assim espaço para inovação neste problema. Os métodos apresentados neste capítulo são antigos e não tratam do problema de balanceamento de carga em questão, abrindo espaço assim para o surgimento de tecnologias emergentes capazes de tratarem o problema com uma outra perspectiva, com uma visão centralizada da topologia da rede e com facilidade de implementação e gerência desses problemas.

Além de todos os fatores ressaltados percebemos que há dificuldade em implementar melhorias nos modelos existentes. Todos em sua maioria são definidos por milhares de linhas de código gerando asim complexidade para serem implementados. Há dificuldade em inovar neste assunto nas redes atuais. Os recursos disponíveis são em sua maioria compostos por *hardwares* proprietários com suas *APIs* próprias atendendo a problemas específicos, além de tornar um desafio para a gerência da rede tendo em vista a sua complexidade através desses modelos. São necessárias mudanças significativas no modo de criação e de operação das redes de computadores, as atuais redes de computadores apresentam diversas demandas que ainda precisam ser discutidas e tratadas. Precisamos contar com os recursos da rede da mesma forma que contamos com os recursos de armazenamento e computação.

3 REDES DEFINIDAS POR SOFTWARE

3.1 INTRODUÇÃO

Redes definidas por *software* (conhecida como SDN, acrônimo do inglês, (*Software Defined Networking*) é uma arquitetura de rede de computadores emergente onde o controle de rede está dissociado do encaminhamento de pacotes e que pode ser diretamente programável (FOUNDATION, 2012). A arquitetura atual de dispositivos de redes (*switches* e roteadores) é baseada em duas camadas distintas: o software de controle (*control path*) e o *hardware* dedicado ao encaminhamento dos pacotes (*datapath*) (NETWORKS, 2010). O software de controle é encarregado de tomar as decisões de roteamento e transferi-las para o plano de encaminhamento através de uma *API* proprietária. A única iteração da gerência com o dispositivo de rede ocorre através de interfaces de configuração (*Web* ou CLI), limitando assim o uso desses dispositivos às funcionalidades programadas pelo fabricante.

A implantação da tecnologia SDN pode ser utilizada para resolver vários tipos de problemas de gerenciamento de rede em redes atuais. Para entender melhor os benefícios de SDN, podemos comparar a arquitetura atual das redes de computadores com a arquitetura SDN. Quando adquirimos um roteador, por exemplo, adquirimos um *hardware* e um *software* proprietário. Na Fig. 3.1 podemos visualizar a arquitetura atual dos dispositivos de redes, separada em partes distintas e realizando funções totalmente separadas.

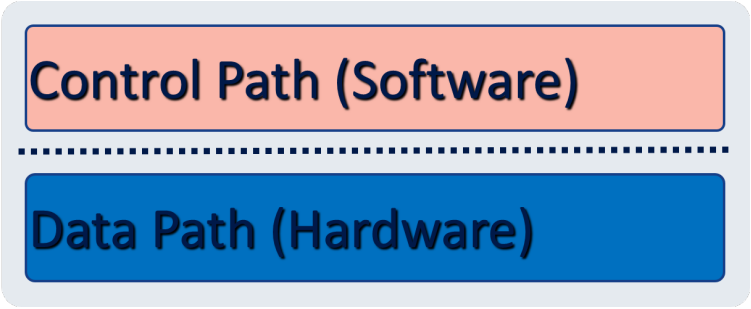


FIG. 3.1: Arquitetura atual dos dispositivos de redes (MCKEOWN, 2010)

Redes definidas por *software* quebra eficazmente essas partes em pedaços. O nível

mais baixo é do encaminhamento de dados, que é responsável por nada mais do que encaminhar o tráfego dos pacotes. Acima temos o nível de controle que é responsável por definir o controle da rede. SDN separa o plano de controle (*control plane*) do plano de encaminhamento (*data plane*) em roteadores e *switches* de rede. A tecnologia de SDN propõe que o plano de controle seja implementado por *softwares* alocados em servidores, e que o plano de encaminhamento esteja nos equipamentos de rede (*switches* e roteadores).

Na tecnologia SDN, o equipamento de rede deve estar habilitado com o protocolo OpenFlow para que o mesmo possa se comunicar com o controlador através de um canal seguro. O equipamento de rede continua possuindo duas camadas distintas (*control plane* e *data plane*), porém estes equipamentos passarão a utilizar somente a camada de encaminhamento já que a camada de controle será diretamente programável através do controlador OpenFlow, o qual definirá o plano de controle. A Fig. 3.2 apresenta essa arquitetura.

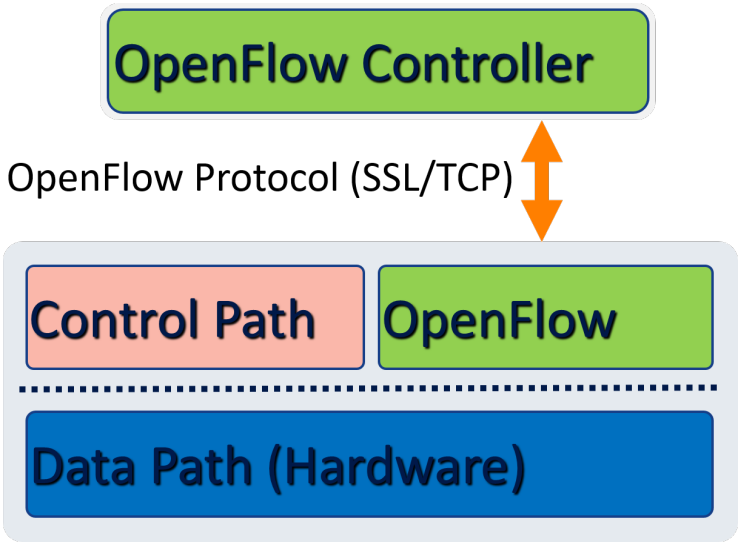


FIG. 3.2: Arquitetura dos dispositivos de redes proposta pelo SDN (MCKEOWN, 2010)

3.2 ORIGEM E HISTÓRIA

A origem da tecnologia SDN foi motivada pela observação dos sistemas de configuração de redes distribuídas, os quais configuram os dispositivos de rede independentemente em equipamentos de baixo nível. A uns 10 anos atrás muito foi feito para se criar uma maneira de confinar esses problemas para que fosse assim possível entender e analisar os

resultados a fim de se propor soluções. Com isso, muitas comunidades se empenharam para aprender, entender e analisar o comportamento da rede a partir desses arquivos de configuração de baixo nível. Enfim, descobriam que a conclusão seria difícil e seria mais fácil se um ponto centralizador pudesse ditar o comportamento de encaminhamento dentro rede. Este conceito é utilizado na Internet pelos protocolos BGP, apresentado no Capítulo 2 deste trabalho, e o RCP (*Routing Control Protocol*). Mais tarde esta arquitetura foi generalizada e chamada de arquitetura 4D. Nesta arquitetura a camada de encaminhamento está no nível mais baixo e é responsável pelo encaminhamento do tráfego na rede. Acima da camada de encaminhamento existiria a camada de descoberta e disseminação. A camada de descoberta permitiria que o ponto de controle da rede pudesse descobrir os recursos disponíveis na rede e disseminá-los para a camada de decisão a qual controlaria a rede RCP. Poucos anos após, surgiu o OpenFlow com um poderoso nível de controle que poderia controlar o comportamento do *switches* e roteadores através de uma interface bem definida. A Fig. 3.3 apresenta a origem da tecnologia de redes definidas por *software*.

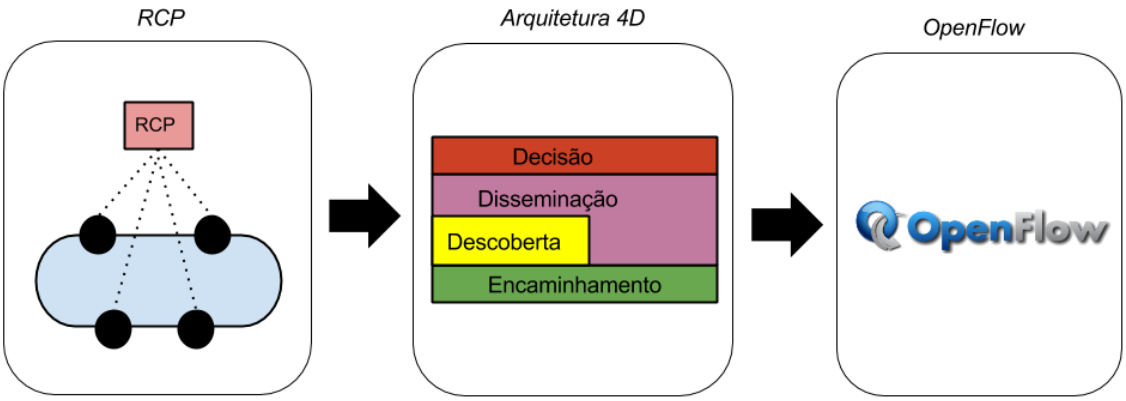


FIG. 3.3: Origem de redes definidas por *software*

3.3 VANTAGENS DE SDN SOBRE REDES ATUAIS

SDN oferece um maior controle da rede através da programação. Este recurso combinado com outras tecnologias provê um potencial de benefícios para melhorias de configuração, melhor desempenho e incentivo a inovação nas operações de rede. Além disso, com a capacidade de obter informações da rede a partir de um controlador centralizado em tempo real baseado nos status da rede e de políticas previamente definidas. Isto nos leva

a melhorias de otimização de desempenho da rede. Nesta seção serão apresentados alguns benefícios de SDN sobre as redes atuais e os fatores que implicam para o surgimento de SDN como tecnologia emergente com inúmeras vantagens sobre as redes atuais:

- Otimizando a configuração: Sabemos que no gerenciamento de rede, as configurações realizadas nos equipamentos de redes são de extrema importância. Especialmente quando novos equipamentos são adicionados ou removidos da topologia da rede. Em redes convencionais é necessário que o administrador da rede configure e controle cada equipamento de rede de forma independente a partir de alguma linguagem de baixo nível de cada fornecedor. Com SDN as diferentes camadas, plano de controle e plano de encaminhamento, são separadas e podem ser controladas a partir de um ponto central na rede. Assim, o comportamento do controle da rede atinge um alto nível de programação onde é possível definir as políticas da rede de um ponto central. Os equipamentos atuais são limitados e possuem *hardware* proprietários o que dificulta o aprendizado e a programação da linguagem específica do fornecedor. SDN fornece uma visão central da rede enquanto os protocolos implementados atualmente possuem uma visão parcial da topologia, o que contribui para a limitação das soluções propostas para os problemas convencionais.

- Melhoria de performance: Nas operações de redes de computadores, um dos objetivos chaves é maximizar a utilização dos recursos da rede. Entretanto esse trabalho tem se tornado difícil, mediante a coexistência de várias tecnologias dentro de uma mesma rede. SDN provê a oportunidade de melhorar a performance da rede de uma maneira global, através de um controle centralizado com uma visão global da rede. Assim problemas convencionais de melhoria de performance podem se tornar mais gerenciáveis adequadamente com algoritmos centralizados.

- Protocolo aberto: Através do protocolo OpenFlow é possível definir uma interface aberta de comunicação entre os dispositivos da rede e ponto central da rede. Ele é um protocolo aberto para troca de mensagens entre os controladores e os equipamentos de redes. Ele usa conceito de fluxos para identificar o tráfego da rede baseado em regras pré-definidas que podem ser programadas estaticamente ou dinamicamente pelo controlador SDN.

- Espaço para inovação: Com SDN, geramos espaço para inovação pois nas redes está cada vez mais difícil inovar. Os protocolos já estão mais que adaptados e

possuem inúmeras linhas de código em sua definição o que implica em protocolos muito ultrapassados e que definem soluções específicas para cada problema da rede. Através de redes definidas por *software* há espaço para a criação e customização de sua própria aplicação, onde se pode resolver problemas generalizados de acordo com a sua necessidade.

3.4 ARQUITETURA SDN

Em uma arquitetura SDN (REITBLATT, 2011), a inteligência da rede é centralizada em controladores baseados em *software* SDN, que mantém uma visão global da rede. O *software* de controle e o *hardware* dedicado ao encaminhamento de pacotes são duas camadas bem distintas, e não precisam estar contidas em um mesmo equipamento. Para isso, basta que exista uma forma padrão de se programar o dispositivo de rede remotamente, permitindo que a camada de controle possa ser movida para um servidor dedicado e com alta capacidade de processamento. Desse modo, mantém-se o alto desempenho no encaminhamento de pacotes em *hardware* aliado à flexibilidade de se inserir, remover e especializar aplicações em *software* por meio de um protocolo aberto para programação da lógica do equipamento (MCKEOWN, 2008).

No modelo das redes definidas por *software* o plano de controle e o de gerenciamento passam a ser diretamente programáveis, obtendo-se assim uma inteligência de rede centralizada podendo abstrair a infraestrutura de rede para o desenvolvimento de novas aplicações separando o plano de controle do plano de encaminhamento.

A Fig. 3.4 retrata a arquitetura da tecnologia SDN. Esta arquitetura é composta em camadas. Na camada inferior, ou camada de infraestrutura, temos os equipamentos de rede com o protocolo OpenFlow (*switches* e roteadores). Esses equipamentos possuem somente o plano de encaminhamento de pacotes e que executam as ações dos controladores. Na camada de controle, temos os controladores baseados em *software* que possuem a visão global da rede e que gerenciam as políticas de encaminhamento de pacotes na arquitetura SDN. Na camada superior, ou camada de aplicação, temos um conjunto de (*APIs*), que torna possível a criação, customização ou implementação dos mais comuns serviços de redes incluindo: roteamento, segurança, controle de acesso, gerenciamento de largura de banda, engenharia de tráfego, balanceamento de carga, qualidade de serviço, entre outros.

SDN possui vários benefícios, as empresas podem administrar o controle da rede in-

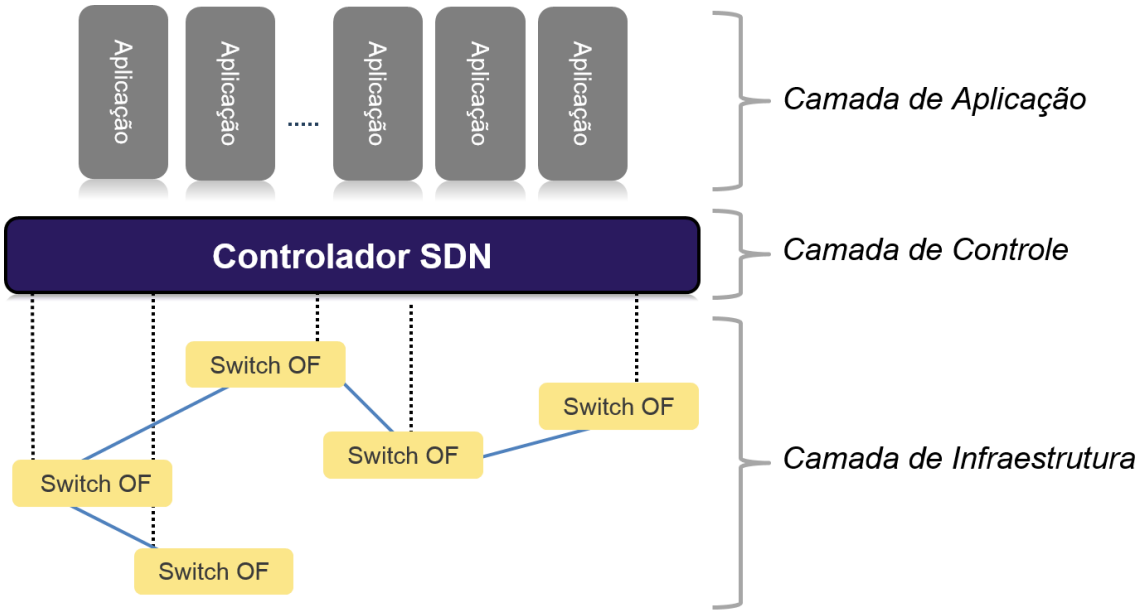


FIG. 3.4: Arquitetura SDN

dependentemente dos seus fornecedores de equipamentos de rede. A partir de um único ponto lógico, a rede passa a ser controlada, o que simplifica o seu desenho e operação uma vez que os equipamentos de redes não precisam entender e processar milhares de padrões de protocolos, basta somente aceitar as instruções dos controladores SDN. Além disso, através da camada de aplicação os operadores e administradores da rede podem configurar programaticamente a rede simplificada ao invés de ter que configurar manualmente os milhares de dispositivos de rede distribuídos na topologia. Para funcionamento do SDN, três fatores são necessários:

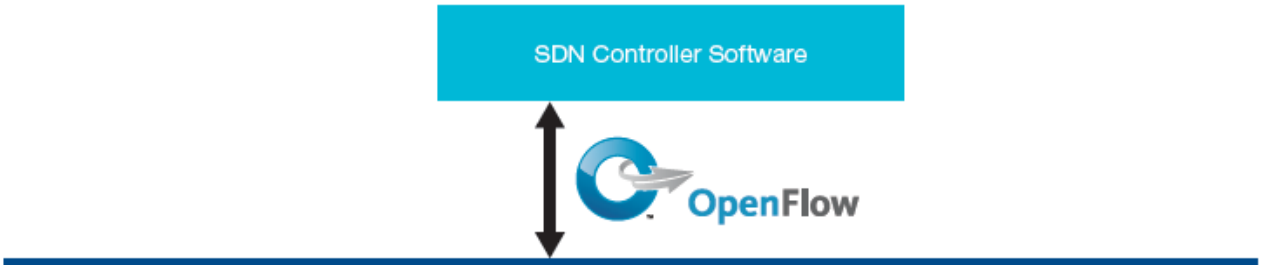
- Uma interface aberta de comunicação entre o controlador SDN e os dispositivos de redes;
- Um sistema operacional (SO) capaz de controlar a rede;
- Aplicações (*APIs*) bem definidas.

A tecnologia SDN promete transformar as redes atuais em plataformas programáveis e flexíveis com inteligência para alocar recursos de forma dinâmica. O SDN está caminhando para se tornar a nova norma das redes atuais.

3.5 OPENFLOW

De acordo com (MCKEOWN, 2008) o protocolo OpenFlow é a primeira interface padrão de comunicação definida entre o controlador e as camadas de encaminhamento de uma arquitetura de redes definidas por *software*. O *Open Networking Foundation* (FOUNDATION, 2012) é responsável pela padronização do protocolo OpenFlow, a fim de garantir a interoperabilidade entre dispositivos de rede e *software* de controle de diferentes fornecedores. OpenFlow já está sendo amplamente adotado por fornecedores de infraestrutura.

Como mostra a Fig. 3.5, o protocolo especifica primitivas básicas que podem ser usadas por uma aplicação de *software* externo para programar o plano de encaminhamento dos dispositivos de redes, assim como o conjunto de instruções em uma CPU que podem programar um sistema de computador.



OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

FIG. 3.5: Conjunto de instruções do protocolo OpenFlow (FOUNDATION, 2012)

O protocolo do OpenFlow é implementando como interface de comunicação entre os dispositivos de rede e a camada SDN de controle. Ele usa um conceito de fluxos para identificar o tráfego da rede baseado em regras pré-definidas que podem ser programadas estaticamente ou dinamicamente pelo controlador SDN. Atuais protocolos de roteamento baseados em IP não permitem esse nível de controle, pois todos os fluxos entre

os dois terminais devem seguir o mesmo caminho através da rede, independentemente das suas diferentes necessidades. O protocolo OpenFlow é implementado em ambos os lados das interfaces entre os dispositivos de redes e os controladores SDN. Com o OpenFlow, pesquisadores podem criar novos algoritmos e rodar seus experimentos através de um protocolo padronizado sem que os fabricantes tenham que expor a implementação interna dos seus equipamentos.

Uma rede programável com OpenFlow (ROTHENBERG, 2010) consiste em equipamentos de rede habilitados para que o estado da tabela de fluxos possa ser instalado através de um canal seguro, conforme as decisões de um controlador SDN:

- Tabela de fluxos - cada entrada na tabela de fluxos do *hardware* de rede consiste em regras, ações e contadores.
- Canal seguro - ele garante a confiabilidade na troca de informações entre o equipamento de rede e o controlador.
- Protocolo OpenFlow - um protocolo aberto para troca de mensagens entre os controladores e os equipamentos de redes.
- Controlador - é o *software* responsável por tomar decisões, adicionar ou remover entradas na tabela de fluxos, de acordo com o objetivo desejado.

O protocolo OpenFlow é um elemento essencial para o conceito de SDN e, atualmente, é o único protocolo padronizado SDN que permite a manipulação direta do plano de encaminhamento de dispositivos de rede. O protocolo OpenFlow encontra-se em sua versão 1.3.2. Neste trabalho usaremos a versão 1.0.0 do protocolo, por se mostrar uma versão mais madura e avaliada nos trabalhos publicados de redes definidas por *software*. A versão 1.4 do protocolo foi aprovada, mas ainda está sob ratificação. E a versão 1.5 está sendo desenvolvida.

Dentro do protocolo OpenFlow os *switches* são chamados de *datapaths*. A tabela de fluxos é onde estão contidas as informações que formam os fluxos em que os pacotes devem ser encaminhados. Além disto ela contém também os contadores e as informações estatísticas a serem executadas e/ou mantidas por um *datapath*. Todos os pacotes que passam por dentro de um *datapath* possuem seu cabeçalho confrontado com a tabela de fluxos.

Na especificação do OpenFlow 1.0.0 (CONSORTIUM) um fluxo é formado pelos campos do cabeçalho, contadores e ações, conforme apresenta a Fig. 3.6 e que são descritos como:

- Campos do cabeçalho: informações encontradas nos cabeçalhos do pacote, porta de entrada e metadados usados para validar os pacotes de entradas;
- Contadores: usados na coleta de estatísticas para um fluxo em particular, como por exemplo, o número de pacotes de um fluxo entrante no *datapath*;
- Ação: que são aplicadas depois de cada validação as quais determinam como lidar com os pacotes correspondentes.

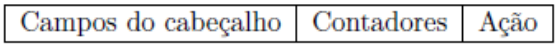


FIG. 3.6: Campos de um fluxo OpenFlow (CONSORTIUM)

O campos do cabeçalho são verificados durante a comparação entre o pacote de entrada e tabela de fluxos. Na versão 1.0.0 do protocolo OpenFlow, ele é apresentado na Fig. 3.7.

Ingress Port	Ethernet src	Ethernet dst	Ethernet type	Vlan ID	Vlan Priority	IP src	IP dst	IP proto	IP TOS	TCP/UDP src port	TCP/UDP dst port

FIG. 3.7: Campos do cabeçalho de um pacote OpenFlow (CONSORTIUM)

Os seguintes campos constituem o cabeçalho de um pacote Openflow:

- *Ingress Port*: Porta de entrada do switch;
- *Ethernet source*: Endereço MAC de origem;
- *Ethernet dst*: Endereço MAC de destino;
- *Ethernet type*: Tipo do quadro (frame) *Ethernet*;
- *Vlan id*: Número de identificação da VLAN (*Virtual LAN*);
- *Vlan priority*: Nível de prioridade;

- *IP src*: Endereço IP de origem;
- *IP dst*: Endereço IP de destino;
- *IP proto*: Protocolo IP;
- *IP ToS* (*Type of Service*): Tipo de serviço;
- *TCP/UDP src port*: Porta de origem do protocolo (TCP/UDP);
- *TCP/UDP dst port*: Porta de destino do protocolo (TCP/UDP).

Os contadores são mantidos por tabela, por fluxo ou por fila. Os contadores podem ser utilizados para a geração de estatísticas da rede. Existem várias aplicações *APIs* que consultam essas estatísticas a fim de monitorar a rede.

Os *datapaths* podem realizar diversas ações com os pacotes correspondentes a um determinado fluxo, sendo que um mesmo pacote pode corresponder a mais de um fluxo, portanto, a mais de uma ação diferente. Essas ações podem ser:

- Encaminhamento (*In Port*): deve permitir o encaminhamento de mensagens para portas físicas ou virtuais.
- Encaminhamento (*Controller*): o pacote é encapsulado em uma mensagem packet-in e é enviada para o controlador.
- Encaminhamento (*Local*): o pacote é enviado em todas as portas físicas ou virtuais, não incluindo a porta de origem.
- Enfileiramento (*Enqueque*): O pacote será encaminhado para uma fila anexada a uma porta que será responsável pelo padrão de encaminhamento dos pacotes.
- Descarte (*Drop*): É a ação *default*, ou seja, caso nenhuma ação seja especificada para um determinado fluxo, os pacotes serão descartados.
- Modificação de campos (*Modify-field*): Esta ação realiza a modificação do cabeçalho de um pacote, como por exemplo modificar o IP de destino de um pacote.

3.6 SWITCHES

Os *switches* em uma rede SDN são representados como dispositivos de redes de encaminhamento básico e acessíveis através de uma interface aberta, pois a lógica de controle dos algoritmos é de responsabilidade do controlador. Em uma rede OpenFlow os switches são chamados de *datapaths* e possuem duas variedades: Puro ou Híbrido. Switches OpenFlow Puro confiam nas decisões de encaminhando do controlador. Os Híbridos suportam o protocolo OpenFlow além dos protocolos tradicionais, por isso, a maioria dos *switches* comerciais disponíveis hoje são híbridos.

A Fig. 3.8 apresenta uma lista atual das implementações de *switches* acompanhada de uma breve descrição, que inclui a linguagem de implementação e a versão suportada do protocolo OpenFlow.

Software	Implementação	Versão
Open vSwitch	C/Python	V1.0
Pantou/Open WRT	C	V1.0
Ofsoftswitch13	C/C++	V1.3
Indigo	C	V1.0

FIG. 3.8: *Switches* que suportam OpenFlow (MENDONCA, 2014)

Através da Fig. 3.9 podemos ter uma ideia dos *switches* comerciais que estão atualmente disponíveis, seus fabricantes e da versão suportada do protocolo OpenFlow que eles implementam.

Fabricante	Modelo do Switch	Versão
Hewllet-Packard	8200zl, 6600, 6200zl, 5400zl e 3500/3500yl	V1.0
Brocade	NetIron CES 2000 Series	V1.0
IBM	RackSwitch G8264	V1.3
NEC	PF5240 PF5820	V1.0
Pronto	3290 e 3780	V1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 e P3920	v1.2

FIG. 3.9: *Switches* OpenFlow (MENDONCA, 2014)

Quando um pacote chega a um *datapath* OpenFlow, os campos do cabeçalho do pacotes são extraídos e comparados com os campos das entradas da tabela de fluxo. Se alguma

validação for feita, o *datapath* aplica o conjunto de instruções apropriado associado com a tabela de fluxo. Caso a validação não seja feita, a ação do *switch* irá depender das instruções definidas na entrada da *table-miss*. Cada tabela de fluxo deve conter uma entrada na *table-miss* com o intuito de manusear os registros não encontrados nas tabelas. Essas entradas particulares especificam um conjunto de ações a serem realizadas quando nenhuma validação é encontrada para um determinado pacote. Essas políticas incluem ações como descartar um pacote ou encaminhar o pacote diretamente para o controlador através de um canal seguro.

3.7 CONTROLADORES

Um controlador SDN oferece uma interface programável para a rede, onde as aplicações podem ser escritas para realizarem tarefas de gerenciamento e oferecer novas funcionalidades. O controlador possui uma visão centralizada da rede a qual simplifica a aplicação de políticas e o gerenciamento de tarefas entre o controlador e os dispositivos de redes. Dentro de uma rede definida por *software* o controlador é o elemento responsável pela construção e manutenção dos fluxos a serem executados pelos *datapaths*.

Redes SDN possuem planos de controle centralizados ou distribuídos. Um controlador centralizado fisicamente representa um único ponto de falha para toda a rede, portanto o OpenFlow permite a conexão de vários controladores a um *switch*, o que permitiria que controladores *backup* assumissem o controle da rede caso ocorra um evento de falha. Um controlador logicamente distribuído pode ser o resultado de uma inter-rede que abrange vários domínios de controle onde esses domínios podem não concordar com o controle centralizado.

Um controlador SDN pode possuir um modelo de controle reativo ou proativo. Em um modelo reativo, os dispositivos de redes devem consultar o controlador cada vez que uma decisão deve ser tomada, como quando um novo fluxo de pacotes chega ao *switch*. No modelo proativo regras de políticas são publicadas do controlador para os *datapaths*. A Fig. 3.10 apresenta as atuais implementações de controladores disponíveis que suportam o protocolo OpenFlow.

A partir do protocolo OpenFlow, um conjunto de mensagens podem ser trocadas entre o *switch* e o controlador em uma arquitetura SDN. Um controlador pode adicionar, remover ou atualizar uma entrada de fluxo na tabela de fluxo de um *switch*. A Fig. 3.11 demonstra a aplicação de um conjunto de instruções na tabela de fluxos de um *switch*

OpenFlow.

Controlador	Implementação	Open Source	Desenvolvedor
POX	Python	Sim	Nicira
NOX	Python/C++	Sim	Nicira
MUL	C	Sim	Kulcloud
Maestro	Java	Sim	Rice University
Trema	Ruby/C	Sim	NEC
Beacon	Java	Sim	Stanford
Jaxon	Java	SIM	Desenvolvedores Independentes
Helios	C	Não	NEC
Floodlight	Java	Sim	BigSwitch
SNAC	C++	Não	Nicira
Ryu	Python	Sim	NTT, Grupo OSRG
NodeFlow	JavaScript	Sim	Desenvolvedores Independentes
ovs-controller	C	Sim	Desenvolvedores Independentes
Flowvisor	C	Sim	Stanford/Nicira
RouteFlow	C++	Sim	CPQD

FIG. 3.10: Controladores OpenFlow (MENDONCA, 2014)

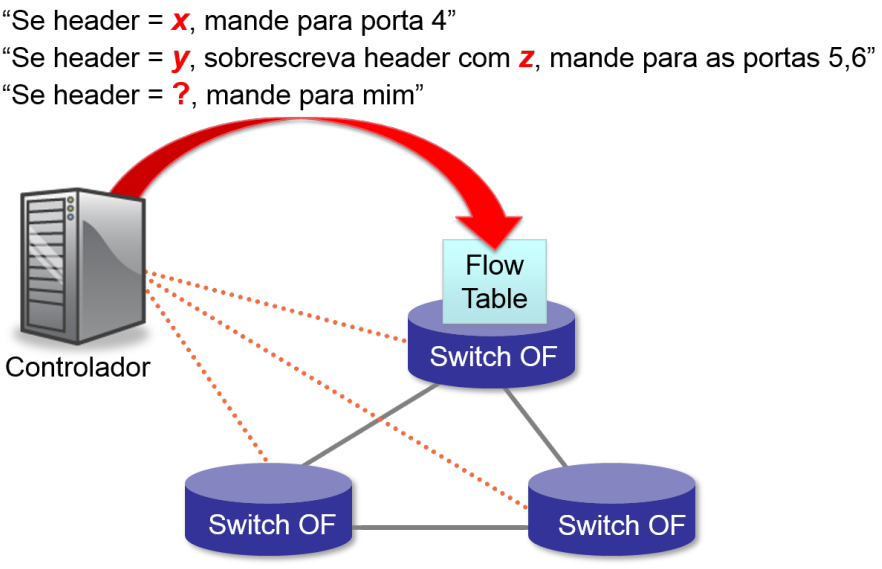


FIG. 3.11: Aplicação de um conjunto de instruções

3.8 FLOODLIGHT

O controlador Floodlight (Projeto Floodlight, 2014) surgiu do controlador Beacon e é mantido, testado e apoiado pela empresa Big Switch Networks e pela maior comunidade de desenvolvedores do mundo para controladores SDN. Ele foi projetado para ser fácil de configurar, com poucas dependências e apresenta uma interface amigável tanto para o desenvolvedor quanto para o usuário. O controlador Floodlight suporta *switches* físicos e virtuais podendo ainda manipular redes híbridas, simultaneamente. O Floodlight oferece um sistema modular, o que facilita bastante para o desenvolvedor estender e melhorar o código. Além disso, qualquer pessoa pode contribuir com o código, o qual pode ser facilmente obtido diretamente do GitHub. A Fig. 3.12 apresenta uma arquitetura de rede OpenFlow com o controlador Floodlight.

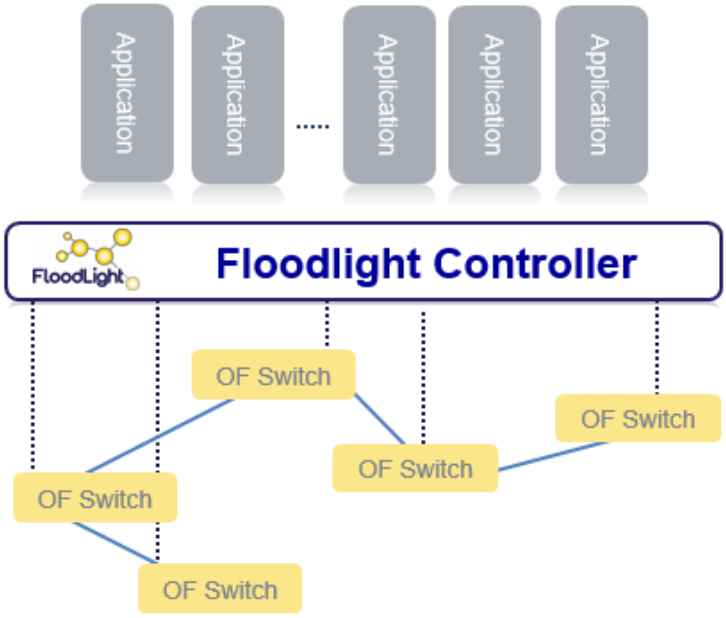


FIG. 3.12: Arquitetura com o controlador Floodlight (FLOODLIGHT)

O controlador Floodlight oferece uma coleção de aplicações desenvolvidas para o mesmo. Estas aplicações fornecem um conjunto de funcionalidades para atender a diferentes necessidades em uma rede OpenFlow. As funcionalidades internas do controlador compreendem um conjunto de serviços e módulos de interesse comum a qualquer aplicação SDN. A Fig. 3.13 ilustra a relação entre o controlador e as aplicações. Os módulos de aplicação são implementados na linguagem *Java* e se comunicam com o controlador através de uma *API* escrita em *Java* . A Fig. 3.13 apresenta a relação entre o

controlador e suas aplicações.

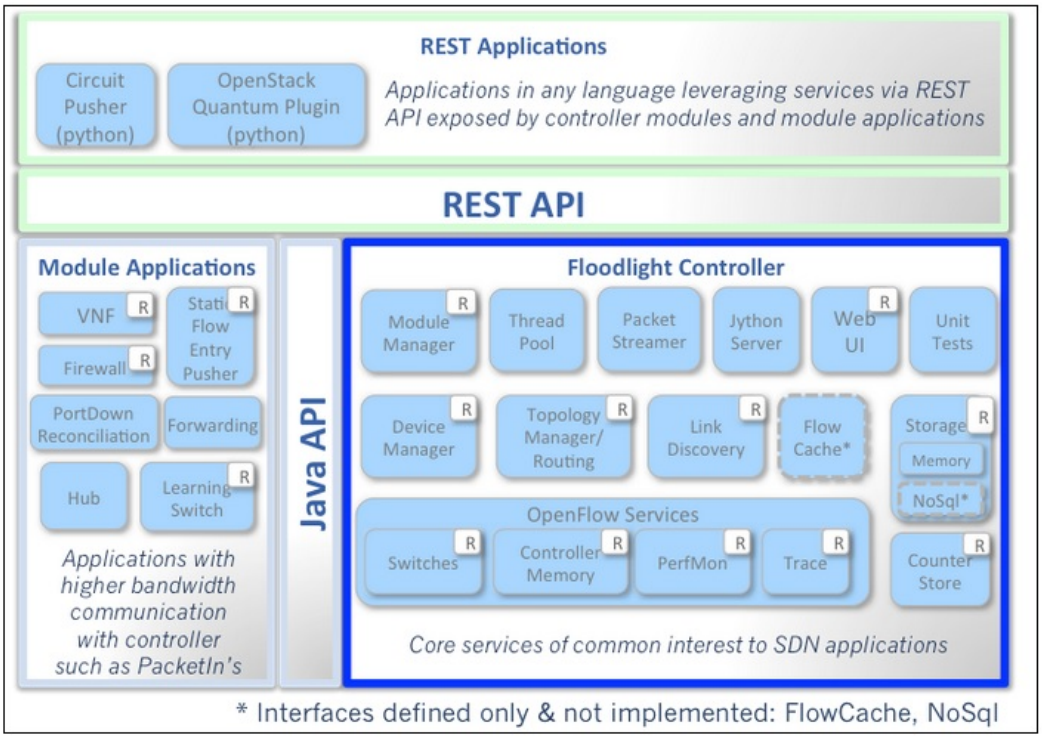


FIG. 3.13: Controlador Floodlight e seus módulos (FLOODLIGHT)

O controlador Floodlight é executado por padrão na porta 6633 e também provê uma interface *web* através da porta 8080, onde é possível consultar informações sobre os *switches* e *hosts*, sobre os fluxos e visualizar a topologia da rede SDN. A Fig. 3.14 mostra o interface *web* e os módulos carregados. O Floodlight foi o controlador escolhido para a validação deste trabalho, pois apresenta inúmeras vantagens, como as apresentadas neste capítulo, sobre os outros controladores SDN.

3.9 MININET

Uma das formas de se construir uma rede virtualizada é a utilizando ferramentas para emulação de redes. Após uma levantamento sobre as ferramentas utilizadas para a validação da tecnologia de redes definidas por *software*, verificamos que há uma ferramenta capaz de simular um *switch* virtualizado e que é bem referenciada em publicações sobre SDN.

Mininet é uma ferramenta capaz de emular uma rede completa de *hosts*, *links* e *switches* em uma única máquina. Ela cria redes virtuais realisticamente dentro de uma única

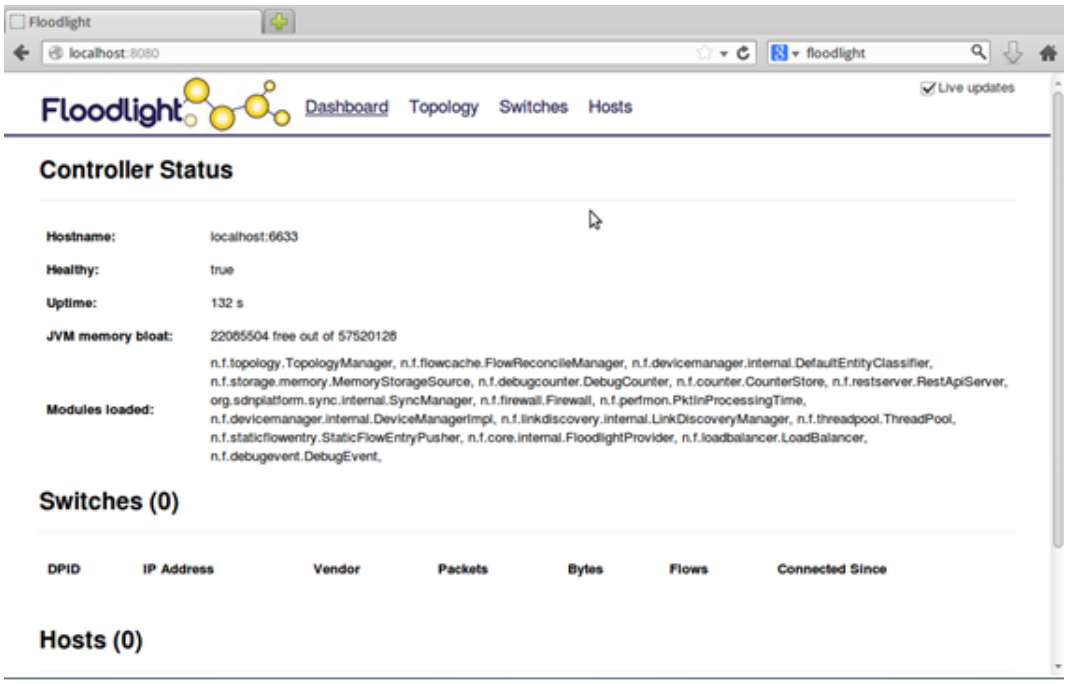


FIG. 3.14: Interface web do Floodlight

máquina virtual contendo um *kernel* e simula *switches* os quais funcionam como switches reais e aplicações de rede. Mininet cria redes virtuais utilizando uma virtualização baseada em processos e *namespaces*, que são recursos recentes disponíveis no *kernel* do sistema operacional Linux. Ela é útil para o desenvolvimento interativo, testes e demonstrações, especialmente aqueles usando OpenFlow e SDN (TEAM, 2014). Algumas características relevantes do Mininet:

- Criar topologias de rede customizadas através de linguagem Python;
- Executar programas reais, como ferramentas de monitoramento como o Wireshark;
- Personalizar o encaminhamento de pacotes através de protocolo OpenFlow;
- Compartilhar e replicar os resultados por ser uma ferramenta *OpenSource*.

O *software* emulador de *switch* utilizado dentro do Mininet é o Open vSwitch. Ele foi criado para poder ser utilizado como um switch comum, com protocolo OpenFlow habilitado, dentro de ambientes virtuais. Os *switches* virtuais OpenFlow criados pelo Mininet fornecem a mesma semântica de entrega de pacotes que seria fornecido por um *switch* físico. Ambos *namespaces* e *kernel* do sistema operacional estão disponíveis. Em

uma emulação Mininet, os controladores podem ser executados na rede real ou emulados, enquanto a máquina onde os *switches* estão em execução tem conectividade com o controlador. Se desejado, o Mininet cria um controlador padrão dentro do ambiente de emulação local e virtual.

Mininet tem sido usado por mais de 100 pesquisadores em mais de 18 instituições, incluindo Princeton, Berkeley, Purdue, ICSI, UMass, Universidade do Alabama Huntsville, NEC, NASA, Deutsche Telekom Labs, Standford e de sete universidades do Brasil. Este fato é um bom indicador das vantagens desta ferramenta.

4 ARQUITETURA PROPOSTA

Neste capítulo apresentaremos a arquitetura proposta neste trabalho para o desenvolvimento do algoritmo de seleção de caminho de custo mínimo para a criação do balanceador de fluxos LBX.

4.1 BALANCEADOR DE FLUXOS LBX

Um dos objetivos deste trabalho é a construção de um aplicação de balanceamento de fluxos, chamada de LBX (Load Balancing X), em uma rede de computadores com suporte a múltiplos caminhos. Nesta aplicação será implementado um algoritmo de roteamento de caminho de custo mínimo, baseado no algoritmo de Dijkstra, onde o intuito é balancear os fluxos da rede baseando-se sempre na seleção do caminho de custo mínimo.

Com os benefícios obtidos através da tecnologia SDN, é possível identificar os fluxos trafegados na rede de uma maneira centralizada. Isso favorece a administração da rede do ponto de vista do administrador da rede e pode contribuir para a diminuição da complexidade na seleção do caminho dentro da rede. No Capítulo 2 deste trabalho, analisamos inúmeros modelos de seleção de caminho dentro de uma rede com suporte a múltiplos caminhos. Vimos que existem modelos que levam em consideração as informações de tráfego dos pacotes e da rede em si. Porém esses modelos não avaliam a rede de uma forma global. As métricas obtidas a partir dessas informações da rede não são vistas e processadas de um ponto central, são informações parcialmente coletadas a partir da visão do nó da rede. A visão global da rede apresenta grandes vantagens que implicam na otimização do balanceamento da carga dentro das redes de computadores. SDN nos favorece com a possibilidade de identificarmos os fluxos trafegados dentro das redes. Como visto, alguns métodos baseados nas informações do pacote como HT, TH e DH executam uma determinada função *hash* com a informação obtida através do cabeçalho do pacote e, a partir desse valor, é definida a seleção do caminho. Com o balanceamento dos fluxos nas redes SDN não é necessária a reordenação dos pacotes, pois o fluxo não é divido em unidades menores, como apresentados em outros médotos, o que é uma grande preocupação para alguns métodos avaliados no Capítulo 2. SDN nos provê esta facilidade de identificarmos os fluxos através do cabeçalho do pacote OpenFlow e realizarmos a seleção do caminho a

partir dessa informação. Não é necessário assim dividirmos os fluxos em unidades menores a fim de encaminharmos cada unidade de fluxo por um caminho distinto.

Através da visão centralizada temos o total controle da administração desses fluxos e podemos monitorá-los assim de forma clara. Todos os caminhos de custo mínimo calculados serão armazenados em um banco de dados da aplicação LBX. Essas informações poderão ser monitoradas a fim de serem reutilizadas para recalcular um novo caminho disjunto de custo mínimo na rede. Com isso, utilizaremos caminhos disjuntos sem interseção de enlaces para o balanceamento dos fluxos. Assim poderemos utilizar todo o recurso da rede, e os nós não ficarão sobrecarregados ou subutilizados. A partir disto, os fluxos serão balanceados pelo enlaces de maneira justa, respeitando a necessidade das aplicações.

Tecnologias baseadas no protocolo OpenFlow permitem enfrentar problemas relacionados à dinâmica necessidade das aplicações atuais, adaptação das redes a cada mudança dos negócios e redução da complexidade no gerenciamento dessas aplicações. Assim sendo, os sistemas de gerenciamento de redes passam a ser adaptados a esta visão centralizada da rede, a qual esta tecnologia nos fornece, implicando diretamente na funcionalidade dos algoritmos para seleção de caminho. Com isso, o processamento e o cálculo desses algoritmos passam a ser executados a partir de um ponto lógico da rede, o qual possui uma visão global da topologia.

A Fig. 4.1 apresenta a arquitetura SDN proposta neste trabalho. Podemos observar a criação da aplicação LBX na camada de aplicação que será executada pelo controlador na rede SDN.

Na arquitetura proposta temos os *switches* (*datapaths*) OpenFlow compondo a camada de infraestrutura. Eles serão responsáveis por simplesmente encaminhar os fluxos dentro da rede SDN. Temos um controlador SDN atuando na camada de controle, ele consulta os *datapaths* da rede SDN e aplica as devidas regras de controle estabelecidas pelo balanceador de fluxos LBX. O controlador conhece toda a topologia da rede e possui uma visão global dos *datapaths*. Este conhecimento é importante na execução do algoritmo de seleção de caminho de custo mínimo, pois o algoritmo consulta a rede para obter informações sobre os *datapaths* e seus enlaces. O LBX é uma nova aplicação construída na camada de aplicação da arquitetura SDN. Essa é uma das vantagens da tecnologia SDN, a possibilidade da criação de suas próprias aplicações conforme necessidade. Dentro da aplicação LBX coletamos as mensagens com o formato *packet-in* que são trocadas na comunicação entre os *datapaths* e o controlador da rede SDN. A partir da captura dessas

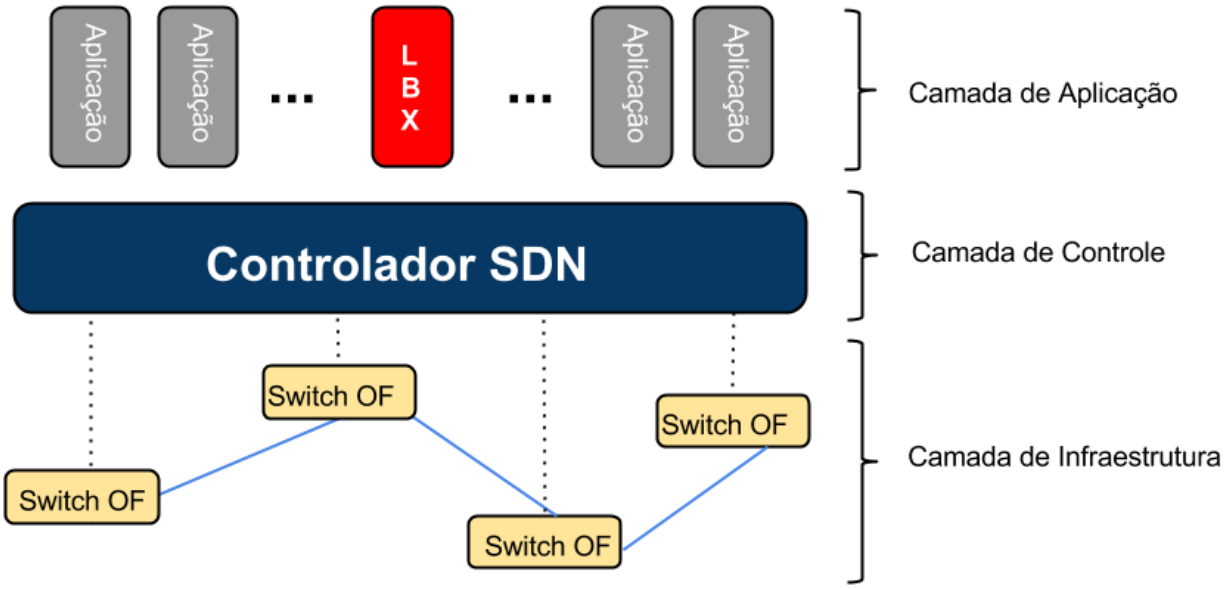


FIG. 4.1: Arquitetura proposta

mensagens são obtidos os identificadores dos fluxos para que haja a separação e o balanceamento dos mesmos dentro da rede SDN. O balanceamento de fluxos nesta arquitetura proposta será obtido sempre através da seleção de caminhos disjuntos que não tenham a mesma intercesão de enlaces. A intensão de usarmos a seleção de caminhos desta forma é para balancear a utilização dos nós, para que assim nenhum enlace ou nó fique sobrecarregado ou ocioso dentro da rede. Assim utilizaremos a capacidade total dos enlaces, garantindo a distribuição dos tráfegos ao longos dos vários caminhos.

Para melhor exemplificar o funcionamento do balanceador de fluxos LBX, definimos um fluxograma o qual apresenta o passo-a-passo das decisões tomadas pela aplicação. A Fig. 4.3 apresenta este fluxograma. No fluxograma é possível identificar a relação do o algoritmo apresentado neste Capítulo com as determinadas ações do sistema.

O fluxograma inicia-se quando um novo fluxo chega ao *switch* e não é identificado na tabela de encaminhamento do *datapath*. O mesmo é encapsulado em uma mensagem no formato *packet-in* e enviado para ao controlador SDN. O controlador entrega essas informações para a aplicação LBX que aciona o algoritmo LBX proposto para verificar no banco de dados da aplicação se existe algum caminho de custo mínimo calculado previamente para determinada origem, destino e identificador do fluxo. As informações de origem, destino e identificador de fluxos são obtidas no cabeçalho da mensagem *packet-*

in trocadas entre os *datapaths* e o controlador. Caso não haja nenhum registro no banco de dados para determinada origem, destino e identificador de fluxo, o método Dijkstra é executado para que seja selecionado o caminho de custo mínimo entre a origem e destino para determinada origem, destino e identificador do fluxo. Realizada a seleção do caminho, o fluxo é enviado pelos *swicthes* pertencentes ao caminho escolhido e a informação sobre o caminho escolhido é devidamente armazenada no banco de dados da aplicação LBX. Caso haja um registro no banco de dados, o método CalculateLBX verifica a possível existência de um outro caminho de menor custo disjunto que não tenha interseção das adjacências selecionadas pelo caminho anterior. Encontrado este caminho, o fluxo é então enviado pelo novo caminho de menor custo e o mesmo também é armazenado no banco de dados da aplicação para possíveis novas consultas. Caso não haja nenhum novo caminho de custo mínimo, o fluxo será enviado pelo caminho já existente, obtido no registro do banco de dados da aplicação LBX. Em todos os casos onde o fluxo seja encaminhado por um determinado caminho de menor custo, SDN nos dá a possibilidade de definirmos ações de um ponto central da topologia e essas ações são replicadas para os *datapaths* da rede. Com isso temos o benefício de atualizarmos automaticamente as tabelas de encaminhamento dos *switches* pertencentes aos caminhos escolhidos, de uma única vez, partindo de uma ação do controlador. Desta forma o algoritmo LBX só será executado uma vez por identificação de um novo fluxo. Realizados os demais passos garantimos o balanceamento dos fluxos através da aplicação LBX em uma topologia SDN, pois a cada fluxo enviado de uma origem para um destino, este é distribuído por um caminho de custo mínimo disjunto calculado a partir da API proposta neste trabalho.

Em uma arquitetura SDN os *switches* são identificados através do seu *datapath ID* (*DPID*). O *datapath ID* é um identificador de caminho de dados definido em uma rede OpenFlow. Ele é um número único de identificação dos *datapaths* dentro de uma rede SDN. Um *switch* pode suportar vários *datapaths*.

No balanceador de fluxos LBX, a fim de identificarmos os fluxos trafegados na rede SDN, utilizaremos o campo identificador do fluxo para diferenciarmos os fluxos dentro da rede, por exemplo, simularmos a separação por determinadas aplicações ou com determinadas prioridades. Os seguintes campos são obtidos dentro da uma mensagem *packet-in* enviada dos *datapaths* para o controlador: *Ingress Port*, *Ethernet src*, *Ethernet dst*, *Ethernet type*, *Vlan id*, *Vlan priority*, *IP src*, *IP dst*, *IP proto*, *IP ToS*, *TCP/UDP src port* e*TCP/UDP dst port*. Esse campos são os mesmos apresentados no cabelhaço de um pa-

cote OpenFlow, no Capítulo 3. Algumas informações do cabeçalho do pacote OpenFlow serão armazenados no banco de dados da aplicação LBX. Os valores dos seguintes campos serão armazenados: Ethernet src (Endereço MAC Origem), Ethernet dst (Endereço MAC Destino) e o IP ToS (Identificador do Fluxo). As informações sobre o *DPID* Origem e o *DPID* Destino também serão armazenadas pois identificam o caminho definido em uma rede OpenFlow. O caminho escolhido também será armazenado. As informações armazenadas poderão ser monitoradas pela aplicação LBX e atualizadas, caso um novo caminho de custo mínimo seja encontrado na rede. As informações sobre Ethernet src (Endereço MAC Origem) e Ethernet dst (Endereço MAC Destino) serão armazenadas simplesmente por já virem no cabeçalho da mensagem OpenFlow. Essas informações não serão utilizadas pelo algoritmo para a seleção do caminho.

Como comentado, para a identificação dos fluxos é utilizado o valor do campo IP ToS do cabeçalho da mensagem OpenFlow. Este campo possui as mesmas condições do campo ToS do protocolo IPv4, os quais são:

- O campo ToS deve ser composto por 8 bits;
- Os 3 primeiros bits indicam o precedence;
- O quarto bit indica quando o atraso é preferido;
- O quinto bit indica quando o *throughput* é preferido;
- O sexto bit indica quando a confiabilidade é preferida;
- E o sétimo e oitavo bits são reservados.

O modelo de serviços diferenciados implementa o *QoS* (em inglês, *Quality of Service*) com base na definição de tipos de serviços. No cabeçalho de um pacote IPv4 em uma rede convencional, existe um campo chamado ToS (*Type of Service*) que pode representar o tipo do serviço. No entanto, serviços diferenciados ampliam a representação de serviços e o tratamento que pode ser dado para encaminhar um pacote, definindo um novo *layout* para o ToS, passando a chamá-lo de *DS Field* (*Differentiated Service Field*). No *DS Field*, são codificadas as classes para serviços diferenciados. O campo ToS já existia na definição do pacote IP. Com DiffServ, os próprios clientes podem marcar seus *DS Fields* e enviar para o receptor. No entanto, dessa forma, não há como saber se há recursos disponíveis para a comunicação, fazendo com que, por exemplo, o pacote chegando em um roteador

que não provê *QoS* com o *DS Field* marcado, seja remarcado e passe a ser um pacote de um serviço de melhor esforço.

Para o balanceador de carga LBX, o campo IP ToS do protocolo Openflow será usado somente para diferenciar os fluxos e não para aplicar nenhuma ação de qualidade de serviço dentro no balanceador. Esse campo será utilizado como suporte a nossa aplicação LBX, pois para a diferenciação dos fluxos será necessário um campo chave e a solução encontrada foi usar o campo IP ToS. As informações contidas no campo IP ToS do cabeçalho OpenFlow podem ser utilizadas para a realização de *QoS* da mesma maneira que são realizadas em redes legadas através do protocolo IPv4. Optamos por utilizar esses valores somente como referência para a identificação dos fluxos na rede SDN. A Fig. 4.2 apresenta os valores do ToS em hexadecimal, que podem ser coletados no cabeçalho da mensagem Openflow e os valores correspondentes ao identificador do fluxo que o balanceador de fluxos LBX usará para identificar seus respectivos fluxos. Esses valores precisarão ser indicados para que possam ser interpretados pela aplicação LBX

TOS (Hexadecimal)	Identificador de Fluxo
0x00	0
0x04	1
0x08	2
0x0C	3
0x10	4
0x20	8
0x28	10
0x30	12
0x38	14
0x40	16
0x48	18
0x50	20
0x58	22
0x60	24
0x68	26
0x70	28
0x78	30
0x80	32
0x88	34
0x90	36
0x98	38
0xA0	40
0xB0	44
0xB8	46
0xC0	48
0xE0	56

FIG. 4.2: Relação ToS e Identificador de Fluxo

4.2 ALGORITMO PROPOSTO

Na API de balanceamento de fluxos LBX, será utilizado um algoritmo proposto para compor a solução de seleção de caminho de custo mínimo em uma rede SDN. O algoritmo será apresentado ao longo deste Capítulo. A aplicação LBX possui a funcionalidade de balancear uma rede SDN através dos fluxos identificados no pacote OpenFlow. Esse balanceamento ocorre de maneira justa pois os caminhos selecionados são armazenados em um banco de dados da aplicação e suas informações podem ser monitoradas pelo algoritmo da aplicação para a seleção de novos caminhos disjuntos. Assim podemos assegurar que todos os caminhos disponíveis na rede serão utilizados afim de atingirmos o balanceamento da carga na rede SDN. O algoritmo proposto neste trabalho é baseado no algoritmo Dijkstra, que calcula o caminho de custo mínimo entre uma determinada origem e destino.

O algoritmo de Dijkstra, cujo nome se origina de seu inventor, o cientista da computação Edsger Dijkstra, calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. O algoritmo considera um conjunto S de menores caminhos, iniciado com um vértice inicial I. A cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S. Repetindo os passos até que todos os vértices alcançáveis por I estejam em S. As arestas que ligam vértices já pertencentes a S são desconsideradas. O Algoritmo proposto neste trabalho é uma adaptação do algoritmo de Dijkstra, onde a verificação da existência de novos caminhos disjuntos faz com que o algoritmo escolha sempre que possível caminhos disjuntos de custo mínimo na rede.

Para um novo fluxo identificado em uma rede SDN, o algoritmo primeiramente identifica a origem, o destino e o identificador do fluxo. A principal característica do algoritmo proposto é que ele possibilita a seleção de novos caminhos de menor custo baseados em caminhos disjuntos os quais não tenham interseção de enlaces. O algoritmo embasa-se em informações armazenadas no banco de dados da aplicação, calculadas previamente pelo próprio algoritmo para assim poder recalcular, caso exista, um novo caminho disjunto de custo mínimo na rede. Quando verifica se existe algum registro no banco de dados para determinada origem, destino e identificador de fluxo, o registro retornado será sempre último da tabela do banco de dados, neste caso o mais atual. Ele calcula este

novo caminho excluindo as adjacências do caminho já existente. Assim podemos garantir que nenhum nó pertencente ao caminho já escolhido será adicionado ao novo caminho. Calculado o novo caminho, o fluxo é então enviado através do novo caminho de menor custo escolhido. As tabelas de encaminhamento dos *switches* que compõem o novo caminho escolhido são devidamente atualizadas pelo controlador da rede. O algoritmo leva em consideração peso padrão 1.0 para todos os enlaces dos nós da topologia SDN. A Fig. 4.4 do Algoritmo LBX apresenta o pseudocódigo do algoritmo implementado na aplicação LBX.

Este algoritmo funciona da seguinte forma. Quando o controlador OpenFlow recebe uma mensagem do tipo *packet-in* de um *switch*, ele passa o controle para o balanceador de fluxos LBX. As linhas 1-3 introduzem a inicialização de variáveis necessárias para a execução do algoritmo. Primeiramente o balanceador de fluxos analisa o cabeçalho do fluxo da mensagem *packet-in* enviada ao controlador. As informações de origem, de destino e do identificador do fluxo são encontradas no cabeçalho da mensagem *packet-in* recebida. Uma vez que essas informações foram obtidas, na linha 4 é realizada uma consulta ao banco de dados da aplicação através do método QueryBD(src, dst, flowID), a fim de encontrar o último caminho de custo mínimo calculado referente a determinada origem, destino e identificador do fluxo. Caso exista algum caminho de custo mínimo previamente calculado, é retornado do banco de dados o último registro referente ao caminho de custo mínimo calculado. Esta ação é necessária a fim de obtermos a informação mais atual sobre os fluxos trafegados na rede SDN. Na linha 5 é iniciada uma estrutura de condição onde, caso a variável path seja diferente de nula, ou seja, caso já exista um caminho de mínimo previamente calculado para determinada origem, destino e identificador do fluxo, um novo caminho de custo mínimo poderá ser proposto pelo método CalculateLBX(path), passando a variável path por parâmetro como mostra a linha 6. Caso contrário, o caminho de custo mínimo será obtido através do método CalculateDijkstra() na linha 7.

A Fig. 4.5 apresenta o pseudocódigo do método CalculateLBX(path) na aplicação LBX. O algoritmo deste trabalho é baseado no algoritmo de Dijkstra, que é o algoritmo que calcula o caminho de custo mínimo entre vértices de um grafo. No método CalculateLBX(path) a variável path é passada por parâmetro, pois ela contém as adjacências escolhidas previamente para o caminho. Na lógica do método CalculateLBX as adjacências já calculadas anteriormente serão excluídas do grafo, para que assim possamos selecionar um novo caminho disjunto no grafo G.

As variáveis de entrada são o gráfico G=(V,E) onde *V* é a classe *Vector* e *E* são os *Edges*. O peso de cada *edge* é definido na variável *edgeW*, o peso do nó é definido pela variável *nodeW* e a nó de origem pela variável *src*. A distância entre o caminho de custo mínimo do nó de origem para o nó *u* de destino são armazenados no vetor *distance[u]*. No vetor *p[u]* são armazenadas os precedentes anteriores a *u* no atual caminho de custo mínimo. Na linha 1, as variáveis são inicializadas pela seguinte forma: distance[s]=0, distance[u]$\in \infty$ para cada u $\in$ V, u$\neq$s. Na linha 2, o vetor priority queue PQ é preenchido com cada nó *u* pertencente ao vetor *V*. Na linha 3, as adjacências calculadas no caminho calculado anteriormente são retiradas do vetor *priority queue* para que assim o novo caminho selecionado seja disjunto ao anterior e não tenha a interseção de enlaces. Na linha 4, enquanto o vetor priority queue PQ for diferente de vazio, as linhas 6, 7, 8 e 9 serão executadas. A partir da linha 5, cada nó pertencente ao grafo será analisada calculando o caminho de custo mínimo para duas adjacências. Lembrando que neste algoritmo todos os pesos dos enlaces possuem custo 1.0. A saída do método CalculateLBX é um vetor contendo o novo caminho de custo mínimo. Com isso, o algoritmo LBX é capaz de retornar um novo caminho de custo mínimo de uma determinada origem para todos os outros nós da topologia, baseado na remoção das adjacências previamente calculadas no caminho anterior.

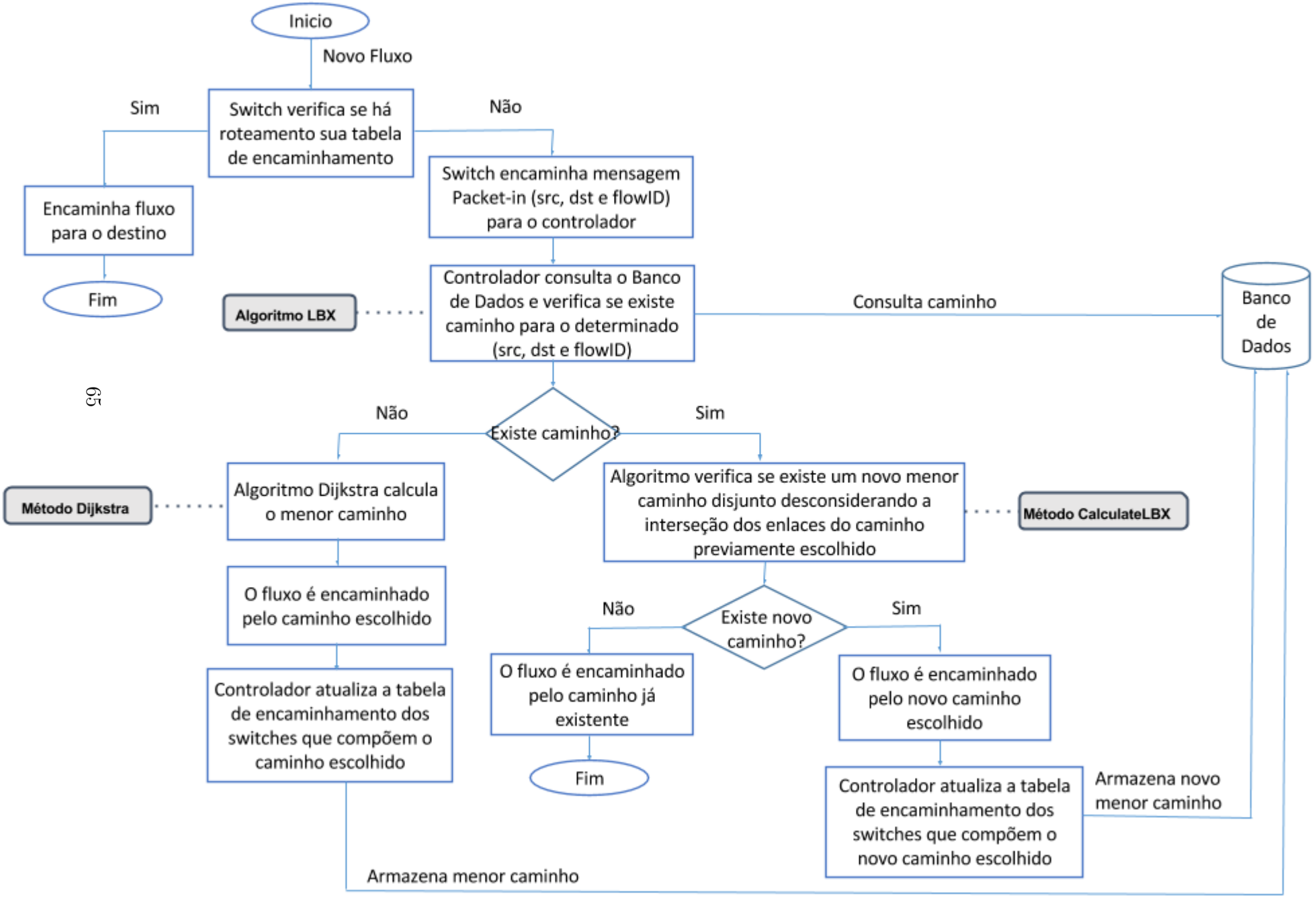


FIG. 4.3: Fluxograma da Proposta

Algoritmo LBX

```
1: src = packet-in (flow);
2: dst = packet-in (flow);
3: flowID = packet-in (flow);
4: path = QueryDB(src,dst,flowID);
5: If path ≠ null  then
6:   newPath = CalculateLBX(path);
7: Else CalculateDijkstra();
```

FIG. 4.4: Algoritmo LBX

Método CalculateLBX(path)

Input: G=(V,E), edgeW, nodeW, src and adjRemove=path;
Ouput: distance[V]

```
1: distance[source] ← 0; distance[u] ← ∞;  para cada u ≠ 0  , u ∈ V
2: insert u with key d[u] into the priority queue PQ, for each u ∈ V
3: remove adjRemove into the priority queue PQ, for each u ∈ V
4:  while (PQ ≠ 0)
5:    u←ExtractMin(PQ)
6:    for each v adjacent to u
7:        if distance[v] > distance[u]+edgeW[u,v]+1.0 then
8:           distance[v] ← distance[u]+edgeW[u,v]+1.0
9:           p[v] ← distance[u]
```

FIG. 4.5: Método CalculateLBX

5 IMPLEMENTAÇÃO

5.1 FERRAMENTAS UTILIZADAS

Com o intuito de viabilizar a implementação da arquitetura proposta, foram utilizadas algumas ferramentas que serão apresentadas neste Capítulo. Durante o processo de escolha das ferramentas, vimos que a maioria delas precisaram ser previamente testadas para que fosse verificado se as mesmas atenderiam à aplicação SDN. A própria tecnologia SDN precisou ser validada inicialmente para verificarmos se realmente o que estava sendo proposto pela tecnologia de fato se concretizaria. A utilização de ambientes virtuais é uma forte tendência para os testes com a tecnologia SDN. Com a dificuldade em se obter equipamentos de rede homologados para o protocolo OpenFlow, um ambiente virtual tornou-se propício para a emulação das redes de testes da tecnologia SDN. A Fig. 5.1 ilustra a arquitetura de implementação proposta.

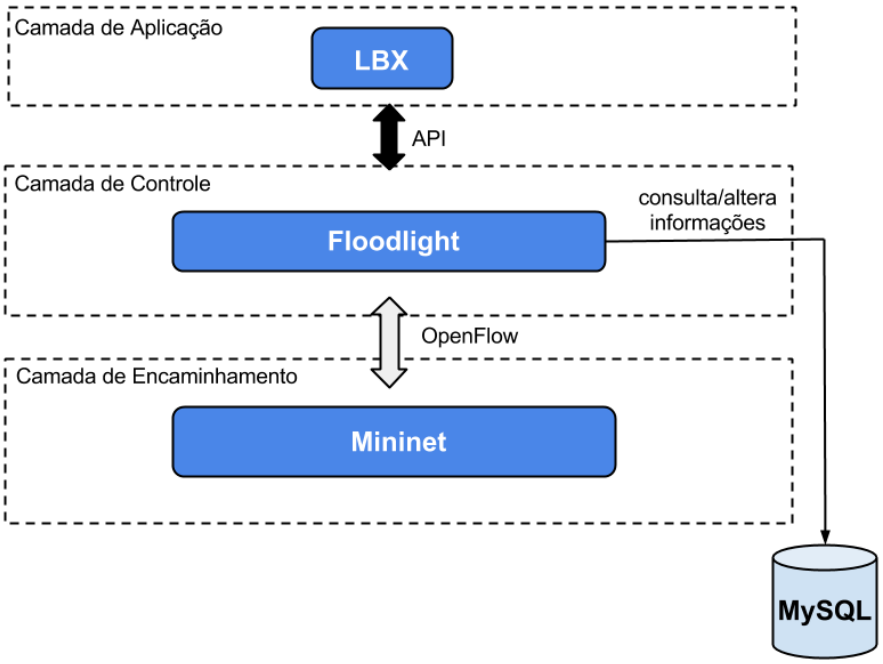


FIG. 5.1: Arquitetura da Implementação

Na camada de encaminhamento, realizaremos a emulação dos *switches* e *hosts* utilizando a ferramenta Mininet, por ser um emulador de redes virtualizadas bem referenci-

ado em outros trabalhos sobre SDN e de grande aceitação na comunidade do OpenFlow. Através do Mininet poderemos emular os *switches* e os *hosts* da topologia SDN. Através da interface CLI do Mininet é possível interagir facilmente com a rede SDN, personalizá-la, compartilhá-la, ou simplesmente implantá-la em um *hardware* real. Mininet é útil para o desenvolvimento, ensino e pesquisa de redes SDN. Com o Mininet temos a facilidade da criação de topologias customizadas através da linguagem Python. As topologias podem ser facilmente customizadas e criadas de acordo com a necessidade de validação. A maneira mais fácil de começar a utilizar o Mininet é realizar o *download* de uma máquina virtual (MV) com todos os binários e ferramentas pré-instaladas OpenFlow, e ajustes para a configuração do *kernel* para suportar redes maiores no Mininet. Também é possível realizar uma instalação nativa a partir do código fonte da ferramenta. Para a implementação deste trabalho o *software* Mininet foi baixado nativamente a partir do código fonte, instalado e configurado na mesma máquina física que o controlador escolhido. Inicialmente o Mininet foi instalado e configurado em uma máquina virtual distinta do controlador, mas por questões de performance decidimos colocar as duas ferramentas na mesma máquina física.

Na camada de controle, utilizaremos o controlador Floodlight, da empresa BigSwitches Networks. A versão instalada foi a 0.90. O Floodlight é implementado na linguagem Java e possui licença *OpenSource*. Ele é constituído por inúmeros módulos, o que facilita a criação e adição de novos módulos, como será realizado com a aplicação LBX. O Floodlight foi escolhido dentre os demais controladores, pois durante os testes mostrou-se um controlador bem robusto e maduro em relação aos demais também analisados (POX e NOX). Ele possui um fórum de *FAQ* bastante ativo e também é muito bem recomendado em pesquisas referentes à tecnologia SDN. Possui uma documentação disponível na Internet bem explicativa e atualizada. O Floodlight suporta topologias com *loops* e possui uma interface *GUI* bem apresentável, chamada de *Avior*. Ele é construído e distribuído pelo *Apache Ant* que permite que você facilmente exporte *JARs* em um sistema de carregamento de módulos. Durante a fase de implementação de trabalho o controlador Floodlight foi o controlador com mais referenciamento positivo encontrado nas publicações sobre a tecnologia SDN. Mostrou-se um controlador bem maduro e foi escolhido por ser facilmente implementado através da linguagem Java. O controlador foi instalado e configurado na mesma máquina física que o Mininet por questões de performance.

Na camada de aplicação temos a nossa a aplicação LBX a qual foi construída através

da linguagem de programação Java pelo algoritmo proposto no Capítulo 4 deste trabalho. A aplicação LBX foi adicionada ao controlador Floodlight como um novo módulo. Um controlador SDN fornece essa possibilidade de criação de novas aplicações para integrarem diretamente como funcionalidade do controlador. Para o desenvolvimento do balanceador de fluxos LBX utilizamos a *API* OpenFlow Java 1.0.1. Foi necessário um estudo prévio sobre a *API* OpenFlow Java para que o algortimo proposto pudesse ser implementado no controlador Floodlight. A *API* OpenFlow Java é uma *API* bem documentada que contempla todos as classes, interfaces, métodos e atributos do protocolo OpenFlow versão 1.0.0 para ser programado na linguagem Java.

O Sistema Gerenciador de Banco de Dados (SGBD) escolhido foi o *MySql* versão 5.5. Ele utiliza a linguagem SQL (do inglês *Structured Query Language*) como interface. É um dos bancos de dados mais populares e possui excelente desempenho e produtividade. Possui licença *OpenSource*, é de fácil programação e é nativamente instalado no *GNU/Linux*. No SGBD foi criada uma tabela chamada lbx contendo os respectivos campos:

- *SwSrcID*: serão armazenados os *datapaths IDs* para cada origem.
- *SwDstID*: serão armazenados os *datapaths IDs* para cada destino.
- *SrcMAC*: serão armazenados os endereços MAC para cada origem.
- *DstMAC*: serão armazenados os endereços MAC para cada destino.
- *FlowID*: serão armazenados os identificadores de fluxo.
- *Path*: serão armazenados os caminhos de custo mínimo para determinada origem e destino.

O algoritmo proposto no Capítulo 4 deste trabalho consulta a tabela lbx do banco de dados *Intel*Mysql da aplicação. Os campos *SwSrcID* e *SwSrcID* serão utilizados para armazenar os respectivos *datapaths IDs* de cada origem e destino do fluxo. Os campos de *SrcMAC* e *DstMAC* são campos obtidos do cabeçalho do fluxo OpenFlow e serão armazenados na tabela sem um motivo específico, eles não são consultados pelo algoritmo durante a sua execução. O campo *FlowID* armazena o identificador do fluxo obtido a partir do cabeçalho da mensagem OpenFlow e o campo *Path* armazena os caminhos de custo mínimo para determinada origem e destino calculados pelo algoritmo LBX, em formato de texto;

5.2 AMBIENTE DE TESTE

Para validar este trabalho, a arquitetura proposta foi implantada em um ambiente físico. O ambiente de teste foi preparado utilizando um computador *Intel* i7 com 4GB de memória *RAM* com o Sistema Operacional Ubuntu 14-02 LTS. Na mesma máquina física foram instalados o controlador Floodlight, o SGBD MySql e o Mininet. Os *hosts* e os *switches* virtuais são emulados através do Mininet. Ele usa um mecanismo de *containers* o qual utiliza grupos de processos que são executados no mesmo *kernel*, além de utilizar recursos separados do sistema, como interfaces de redes e *IDs*. Com isso, os *switches* ou *hosts* emulados possuem o seu próprio processo. As topologias validadas foram criadas a partir de códigos Python e reconhecidas pelo Mininet.

Para a validação do balanceador de carga LBX é necessário a geração de tráfego na rede SDN para que assim os fluxos possam ser identificados e distribuídos aos longos dos caminhos de custo mínimo da rede. Para isso foi necessário realizar um levantamento de possíveis ferramentas para a geração de tráfego na rede SDN. Após a pesquisa decidimos que os fluxos a serem balanceados na rede SDN serão gerados através da ferramenta Iperf, uma ferramenta de teste gratuita capaz de criar tráfegos TCP e UDP entre uma origem e um destino do tipo cliente/servidor desenvolvido pelo *National Laboratory for Applied Network Research* (NLANR). A partir dessa ferramenta poderemos testar/medir o *throughput* da rede (unidirecional e bidirecional) e a perda de pacotes, o *packet-loss*. O Iperf é uma ferramenta frequentemente usada em trabalhos relacionados com a tecnologia de redes definidas por *software*.

6 RESULTADOS

Neste capítulo serão apresentados os cenários utilizados para a validação dessa proposta juntamente com os experimentos realizados neste trabalho.

6.1 CENÁRIOS UTILIZADOS

Neste trabalho foram elaborados dois cenários distintos (C_1 e C_2) para a validação do funcionamento do balanceador de fluxos LBX. O cenário C_1 é um cenário de validação, como mostra a Fig. 6.1, o qual apresenta uma topologia mínima de validação com 3 *switches*, com duas possíveis escolhas de caminho para cada origem e destino. A partir deste cenário podemos balancear os fluxos pelos dois caminhos mínimos possíveis. Cada enlace da topologia foi configurado com uma banda específica, 100 ms de *delay* e 1 % de *packet-loss*. Durantes os testes, foram realizados 5 experimentos distintos, cada um com a sua particularidade, a partir de vários fluxos gerados da origem *Switch* 1 com destino para o *Switch* 2 a fim de verificarmos a distribuição de fluxo através dos dois possíveis caminhos nesta topologia.

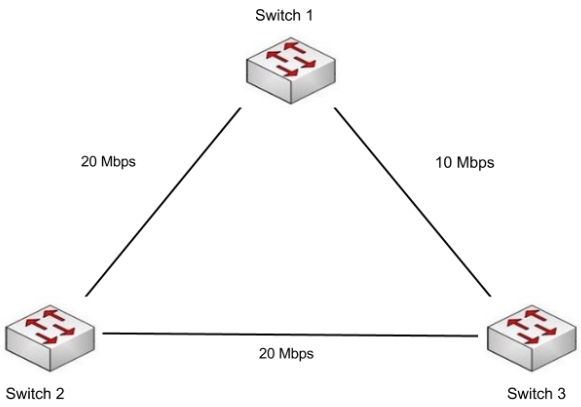


FIG. 6.1: Topologia do cenário C_1

No cenário C_2, temos uma topologia com 6 *switches* com três seleções possíveis escolhas de caminho entre a origem *Switch* 1 e o destino *Switch* 6 como mostra a Fig. 6.2. Em comparação com o cenário C_1, o cenário C_2 possui mais uma opção para seleção de caminho na rede SDN além de possuir mais *switches* na topologia. O cenário C_2 foi uma

adaptação do cenário C_1, onde há mais opções de seleção de caminho de custo mínimo além de mais *switches* na topologia. Assim como o cenário C_1, neste cenário cada enlace foi configurado com a uma banda específica, 100 ms de *delay* e 1 % de *packet-loss*. Para os testes, foram realizados 3 experimentos distintos, a partir de vários fluxos gerados da origem *Switch* 1 com destino para o *Switch* 6 a fim de verificarmos a distribuição de fluxos através dos três possíveis caminhos nesta topologia.

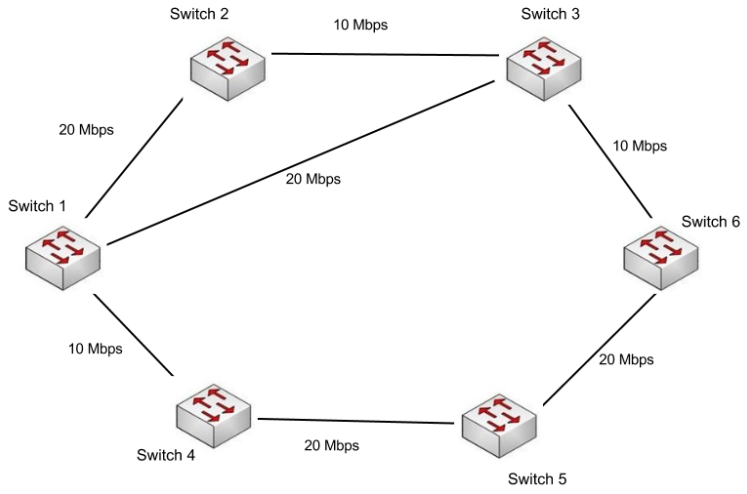


FIG. 6.2: Topologia do cenário C_2

6.2 ANÁLISE DO CENÁRIO C_1

No Cenário C_1 foram realizados 5 experimentos distintos a fim de apresentarmos as métricas de *throughput* e *packet-loss* na rede SDN. O *throughput* representa a quantidade de dados transferidos em um determinado espaço de tempo e o *packet-loss* representa a quantidade de pacotes perdidos em uma transmissão de dados. Para obtermos as métricas de *throughput* e *packet-loss* nos testes, congestionaremos o enlace entre a origem e destino da topologia. Para isto, nos experimentos deste cenário foram enviados 15 fluxos UDP de 1 Mbps e 30 fluxos TCP de 64 bits cada, entre a origem o *Switch* 1 e destino o *Switch* 2. Os fluxos TCP e UDP foram enviados aleatoriamente para assim avaliarmos a distribuição dos fluxos e o desempenho da rede. Notamos que enviando 15 fluxos UDP e 30 fluxos UDP o enlace entre o *Switch* 1 e o *Switch* 2 ficava congestionado. Com isso definimos esta quantidade de fluxos para os experimentos.

6.2.1 EXPERIMENTO 01 - SEM BALANCEAMENTO DE FLUXOS (20 MBPS)

Neste experimento avaliamos a rede SDN sem a realização do balanceamento de carga. É necessário entender como a rede SDN se comporta sem a distribuição dos fluxos pelos caminhos existentes. A banda total disponível para o envio dos fluxos é de 20 Mbps. A capacidade de utilização dos enlaces foi verificada após o envio os fluxos. O experimento teve uma duração de em torno de 10 segundos. Dos fluxos enviados, foi percebido o *throughput* da rede relativamente baixo variando entre de 2.1 Mbps a 2.8 Mbps. A maior taxa de transferência foi identificado a 14.8 Mbps no instante de 6 segundos. A Fig. 6.3 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

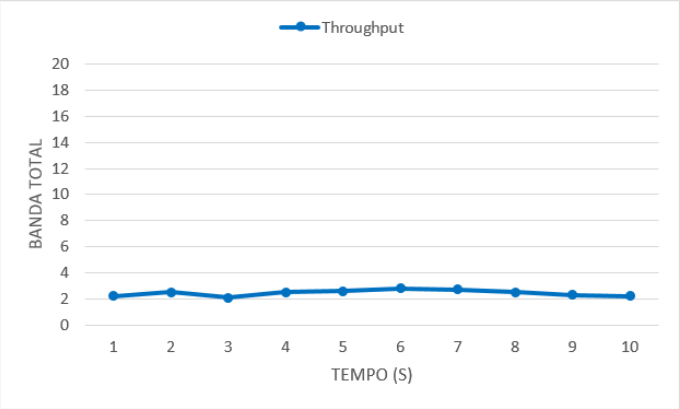


FIG. 6.3: Relatório Iperf para o *Throughput* no Experimento 01 - Cenário C_1

Os testes de *packet-loss* avaliaram a quantidade de pacotes perdidos ao longo do envio dos fluxos entre a origem e o destino na topologia de validação do cenário C_1. A quantidade de perda de pacotes foi avaliada após o envio os fluxos. Através deste teste foi verificado um percentual muito alto de *packet-loss*, o qual variou em torno de 21 % a 50%. A maior perda foi de 50 % no instante de 10 segundos. Através dos resultados podemos avaliar que a falta de balanceamento de carga influencia na baixa taxa de transferência da rede e no alto valor *packet-loss*, pois os fluxos não conseguem ser trafegados por outros caminhos dentro da topologia. Todos os fluxos acabam sendo trafegados por um único caminho. Como o enlace entre o *Switch* 1 e o destino *Switch* 2 foi congestionado, com a falta do balanceamento de carga percebemos que alguns nós da rede ficaram sobrecarregados ou subutilizados. A Fig. 6.4 apresenta os valores de *packet-loss* sem a realização do balanceamento dos fluxos.

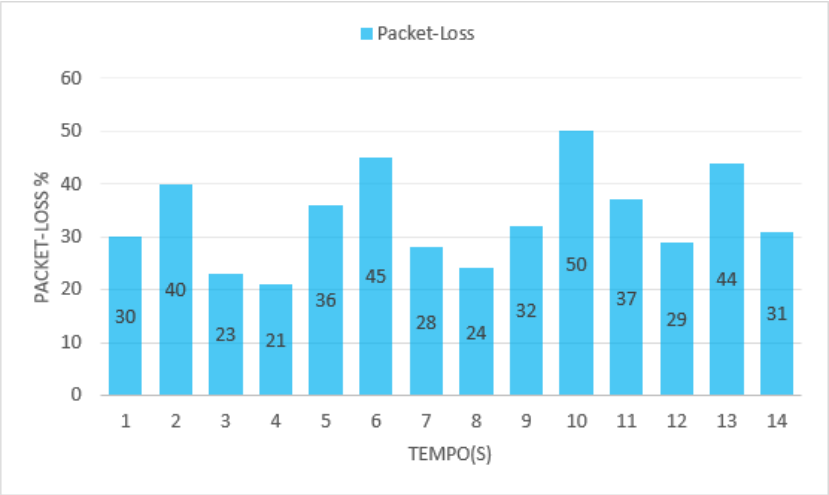


FIG. 6.4: Relatório Iperf para o *Packet-Loss* no Experimento 01 - Cenário C_1

Neste experimento, através do congestionamento dos enlaces verificamos e constatamos o problema para a necessidade do balanceamento de carga. Sem o balanceamento dos fluxos, neste cenário, somente um caminho é escolhido para o envio dos fluxos entre a origem e o destino o que explica a taxa de utilização baixa da rede através dos valores de *throughput* e grande a perda de pacotes relatada pelos altos valores de packet-loss. Este experimento relatou um problema real que podemos tratar com a aplicação do balanceamento de carga, podemos então aplicar a solução LBX proposta neste trabalho.

6.2.2 EXPERIMENTO 02 - COM O BALANCEADOR DE FLUXOS LBX (30 MPBS)

No segundo experimento avaliamos a rede SDN utilizando a aplicação do balanceamento de fluxos LBX. Os testes foram realizados levando em consideração a distribuição dos fluxos pelos possíveis caminhos na topologia. Ao recebimento desses novos fluxos, o *Switch* 1 encaminhará os cabeçalhos dos fluxos para o controlador, que acionará o algoritmo de balanceamento de fluxos LBX. O algoritmo verificará se já existe algum caminho de custo mínimo previamente calculado no banco de dados da aplicação para determinada origem, destino e identificador do fluxo. Caso exista, o algoritmo verificará a possibilidade de existência de um novo caminho disjunto de custo mínimo na rede. Neste experimento foram enviados valores de ToS aleatórios para simularmos fluxos distintos e que os mesmo possam seguir caminhos diferentes na topologia realizando assim o balanceamento dos fluxos. A banda total disponível para o envio dos fluxos passou a ser de 30 Mbps, em vez de 20 Mbps, pois utilizaremos os dois possíveis caminhos para trafegar os fluxos na rede.

Este experimento teve uma duração de 10 segundos. Em comparação ao Experimento 01, realizado sem o balanceamento de carga, é observado que por utilizarmos dois possíveis caminhos para o balanceamento do tráfego, a banda total utilizada no Experimento 02 passa a ser maior. Dos fluxos enviados, foi percebido o *throughput* da rede variando entre de 8.1 Mbps a 8.9 Mbps. A maior taxa de transferência foi identificada a 8.9 Mbps nos instantes de 5 e 10 segundos. Observamos que pela a utilização de mais caminhos para o tráfego dos fluxos, a taxa de transferência da rede foi maior do que comparada ao Experimento 01. A Fig. 6.5 apresenta as informações sobre os valores de *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

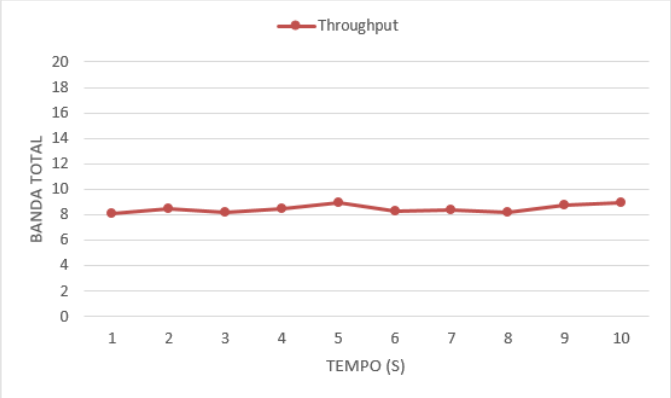


FIG. 6.5: Relatório Iperf para *Throughput* no Experimento 02 - Cenário C_1

O algoritmo LBX armazena cada caminho de custo mínimo selecionado no banco de dados da aplicação. Esses valores armazenados são monitorados na escolha de novos caminhos disjuntos pertencentes a mesma origem, destino e identificador de fluxo. Após a realização deste experimento, verificamos na tabela lbx do banco de dados da aplicação, os fluxos que foram balanceados por cada caminho baseado nas duas possibilidades de caminhos oferecidos na topologia de validação do cenário C_1. Os fluxos com o mesmo identificador foram sempre trafegados pelo mesmo caminho, como mostra a Fig. 6.6. Podemos observar que os fluxos foram identificados pelo algoritmo e distribuído ao longo dos caminhos existentes. O identificador do fluxo é correspondente ao valor definido no campo TOS de pacote OpenFlow, apresentados no Capítulo 4.

Os testes de *packet-loss* apresentaram que os valores variaram de 3.9 % a 5.8 %. A maior perda foi de a 5.8 % no instante de 6 segundos. Notamos que, por haver o balanceamento do fluxos, a perda de pacotes na redes é menor aos valores do Experimento 01. A Fig. 6.7 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do

SwSrcID	SwDstID	SrcMAC	DstMAC	FlowID	Path
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	1	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	2	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	30	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	3	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	1	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	0	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	18	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	2	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	1	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	38	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	48	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	34	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	40	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	14	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	40	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	20	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	28	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	56	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	8	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	32	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	12	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	30	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	26	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	36	1,3,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	1	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	1	1,2
1	2	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:02	48	1,2

FIG. 6.6: Relatório da Tabela lbx - Cenário C_1

Iperf obtidas neste experimento.

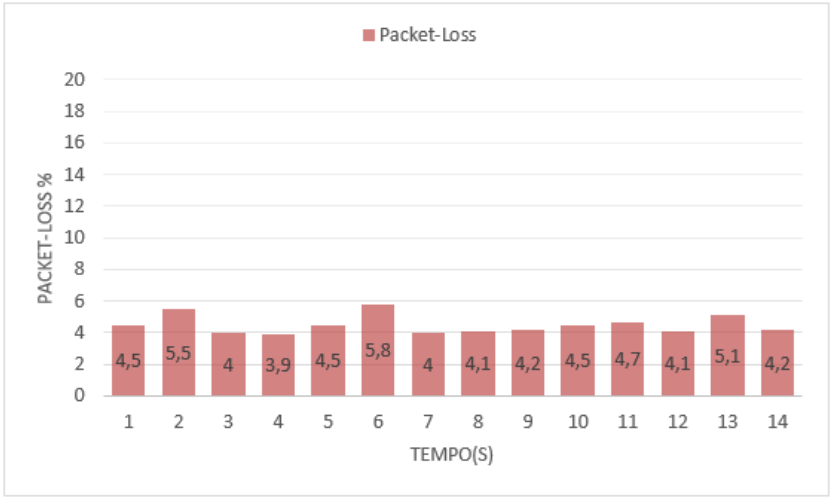


FIG. 6.7: Relatório Iperf para o *Packet-Loss* no Experimento 02 - Cenário C_1

Através da realização do balanceamento dos fluxos é observado que o *throughput* é maior em comparação ao Experimento 01, sem balanceamento de carga. Com o balanceamento dos fluxos há um aproveitamento melhor da banda disponível, pois os fluxos seguem caminhos distintos na topologia, pois estão sendo distribuídos ao longo do vários caminho, evitando congestionamento dos recursos da rede. É observado também que o *packet-loss* é significantemente menor neste experimento, pois através do balanceamento

dos fluxos conseguimos um melhor aproveitamento de rede.

6.2.3 EXPERIMENTO 03 - BALANCEAMENTO POR APLICAÇÃO (30 MPBS)

No terceiro experimento avaliamos o balanceamento de carga simulando o envio de 2 aplicações separadamente. Para isso, foram enviados somente 2 tipos de ToS nos cabeçalhos dos fluxos OpenFlow. Com esse experimento poderemos simular um cenário onde o tráfego dos fluxos é distribuído baseado em duas aplicações diferentes, por exemplo A e B. Assim como no Experimento 02, a banda total disponível para o envio dos fluxos é de 30 Mbps. O experimento teve uma duração de em torno de 10 segundos. Dos fluxos enviados, foi percebido que a taxa de transferência variou de 10.1 Mbps a 10.8 Mbps. A Fig. 6.8 apresenta as informações sobre o *throughput* obtidas neste experimento.

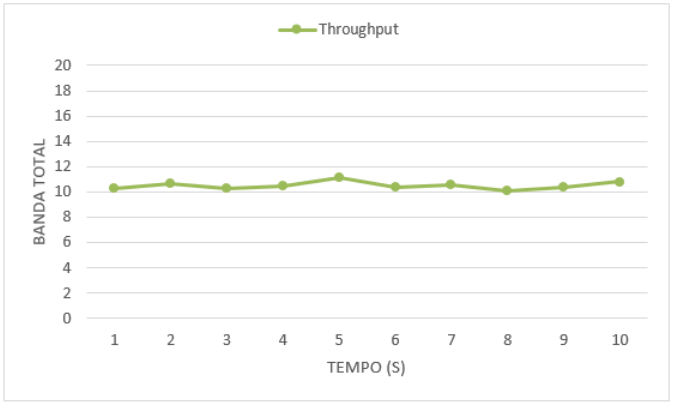


FIG. 6.8: Relatório Iperf para o *Throughput* no Experimento 03 - Cenário C_1

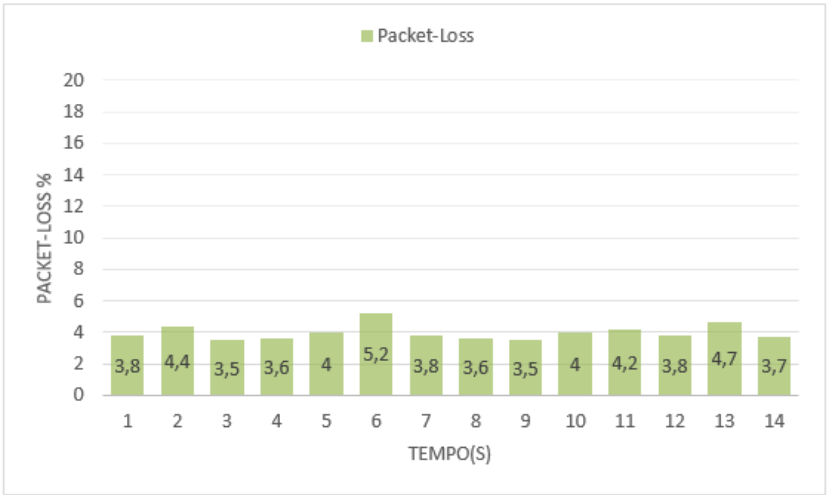


FIG. 6.9: Relatório Iperf para o *Packet-Loss* no Experimento 03 - Cenário C_1

A Fig. 6.9 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do Iperf obtidas neste experimento. Durante os testes de *packet-loss*, foi verificado que o percentual de perda variou de 3.5 % a 4.4 %. A maior perda foi de a 1.3 % no instante de 5 segundos. Em comparação com o *packet-loss* do Experimento 01 e do Experimento 02, a porcentagem de perda do Experimento 03 foi menor, em torno de 1.3 %.

6.2.4 COMPARAÇÃO ENTRE OS EXPERIMENTOS 01, 02 E 03

A partir dos resultados apresentados nos Experimentos 01, 02 e 03 podemos realizar um comparativo dos valores de *throughput* e *packet-loss* no cenário C_1, para assim melhor avaliar a performance da nossa proposta. A Fig. 6.11 apresenta um comparativo sobre os valores de *throughput* entre os Experimentos 01, 02 e 03. A Fig. 6.12 organiza e apresenta um comparativo dos valores obtidos de perda nos Experimentos 01, 02 e 03 para cada instante de tempo.

Throughput			
Intervalo	**Experimento 01**	**Experimento 02**	**Experimento 03**
1	2,2	8,1	10,3
2	2,5	8,5	10,7
3	2,1	8,2	10,3
4	2,5	8,5	10,5
5	2,6	8,9	11,1
6	2,8	8,3	10,4
7	2,7	8,4	10,6
8	2,5	8,2	10,1
9	2,3	8,8	10,4
10	2,2	8,9	10,8

FIG. 6.10: Valores *Throughput* Experimentos 01, 02 e 03 - Cenário C_1

Podemos perceber que os valores de *throughput* do Experimento 01 são bastante inferiores aos dos Experimentos 02 e 03, os quais utilizam a aplicação do balanceamento de fluxos LBX. Nos Experimentos 02 e 03 os índices de *throughput* são maiores do que os valores apresentados no Experimento 01, onde não foi realizado o balanceamento de carga. Notamos que nos experimentos onde há o balanceamento de fluxos LBX, proposto neste trabalho, os valores do *throughput* são melhores mostrando assim que a taxa de transferência da rede SDN é maior com a utilização do balanceamento de carga.

Os valores de *packet-loss* foram bastantes discrepantes em comparação do Experimento 01 aos Experimentos 02 e 03. A Fig. 6.32 apresenta um comparativo sobre os valores

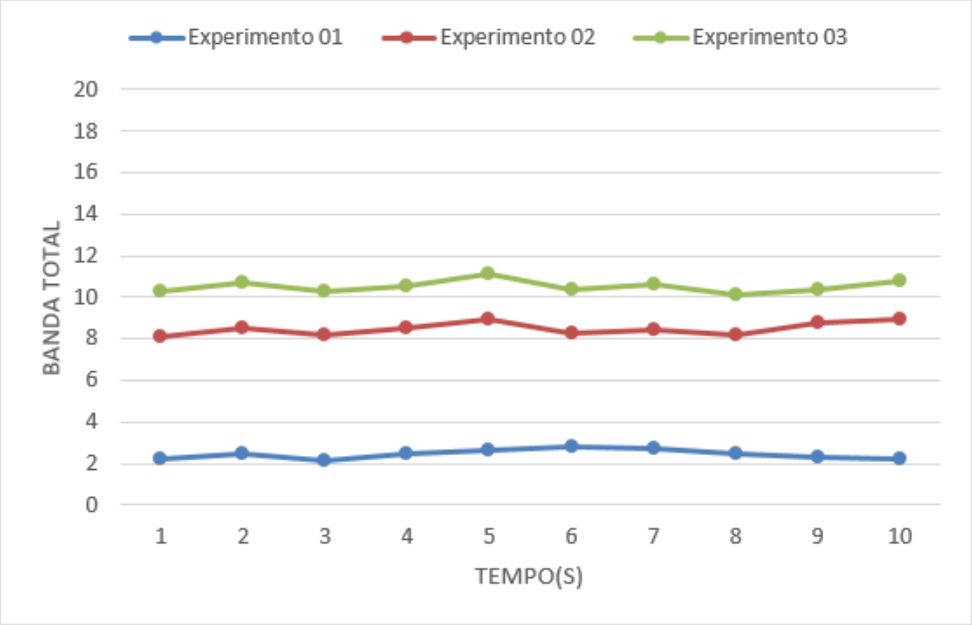


FIG. 6.11: Gráfico Comparativo *Throughput* Experimentos 01, 02 e 03 - Cenário C_1

de *packet-loss* obtidos nos Experimentos 01, 02 e 03. Notamos que a porcentagem de *packet-loss* nos Experimentos 02 e 03, os quais utilizam o balanceamento de fluxo são significantemente menores do que os comparados com o Experimento 01, sem o balanceamento de fluxo. Por não possuir balanceamento de carga, os valores do *packet-loss* são bastante altos em comparação com os Experimentos 02 e 03. Nos experimentos com balanceamento de carga temos uma melhoria significante desse valores mostrando que com o balanceamento ocorre uma diminuição de *packet-loss* e o aumento do *throughput* na rede, tornando assim a rede menos subutilizada e menos congestionada.

Notamos que ao utilizamos o balanceamento de fluxos nos Experimentos 02 e 03 do Cenário C_1, a banda total disponível passa a ser de 30 Mpbs. O que pode ser questionado em comparação com o Experimento 01, o qual não utilizou o balanceamento de carga, porém obteve um valor banda disponível inferior de 20 Mbps, comparada ao Experimento 02. Por isso, achamos necessário repetir os Experimentos 02 e 03 utilizando o mesmo valor de banda total usada no Experimento 01, de 20 Mbps.

6.2.5 EXPERIMENTO 04 - BALANCEADOR DE FLUXOS LBX (20 MPBS)

No quarto experimento avaliamos o balanceamento de carga levando em consideração que os caminhos utilizados tenham o mesmo valor de banda total do Experimento 01.

Packet-Loss			
Intervalo	**Experimento 01**	**Experimento 02**	**Experimento 03**
1	30	4,5	3,8
2	40	5,5	4,4
3	23	4	3,5
4	21	3,9	3,6
5	36	4,5	4
6	45	5,8	5,2
7	28	4	3,8
8	24	4,1	3,6
9	32	4,2	3,5
10	50	4,5	4
11	37	4,7	4,2
12	29	4,1	3,8
13	44	5,1	4,7
14	31	4,2	3,7

FIG. 6.12: Valores *Packet-Loss* Experimentos 01, 02 e 03 - Cenário C_1

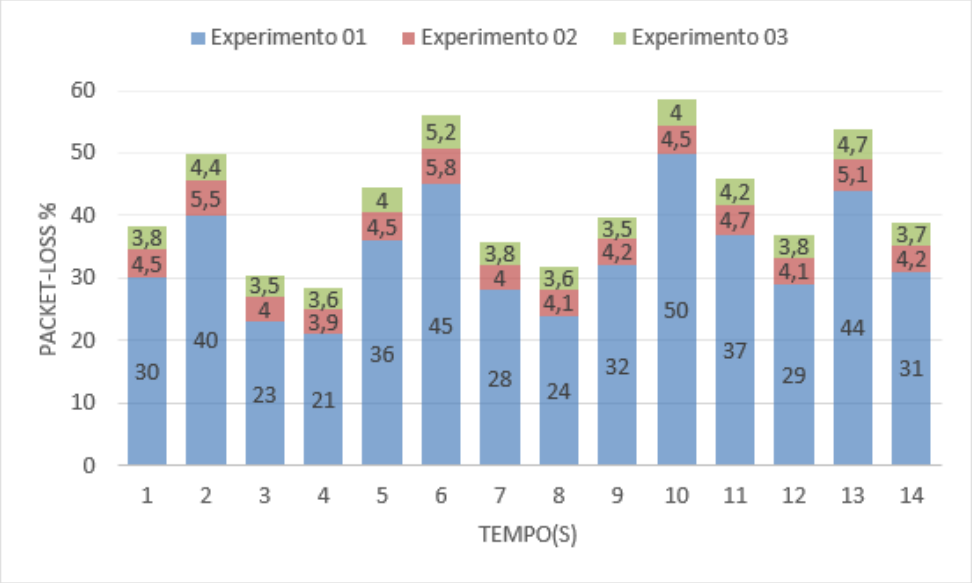


FIG. 6.13: Gráfico Comparativo *Packet-Loss* Experimentos 01, 02 e 03 - Cenário C_1

Analisaremos se os resultados apresentados neste experimento serão diferentes dos valores apresentados no Experimento 02, pois utilizaremos uma banda menor no enlace entre o *Switch* 1 e o *Switch* 2, neste caso 20 Mbps em vez de 30 Mpbs como testado no Experimento 02. Os valores do campo ToS serão enviados também aleatóriamente.

Dos fluxos enviados, foi percebido que o *throughput* da rede variou entre 10.2 Mbps a 10.8 Mbps. A maior taxa de tranferência foi de 10.8 Mbps nos instantes de 5 e 10

segundos. Com a realização do balanceamento dos fluxos nos enlaces com o mesmo valor de banda total utilizado no Experimento 01 foi observado que *throughput* continua sendo maior. A Fig. 6.14 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

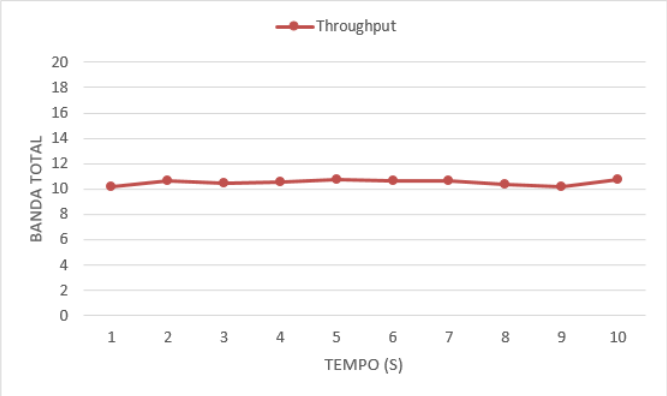


FIG. 6.14: Relatório Iperf para o *Throughput* no Experimento 04 - Cenário C_1

Durante os testes de *packet-loss* também com valores de ToS aleatórios foi verificado que o percentual de *packet-loss* variou de 2.6 % a 4.6 %. A maior perda foi de a 4.6 % no instante de 6 segundos. A Fig. 6.15 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do Iperf obtidas neste teste.

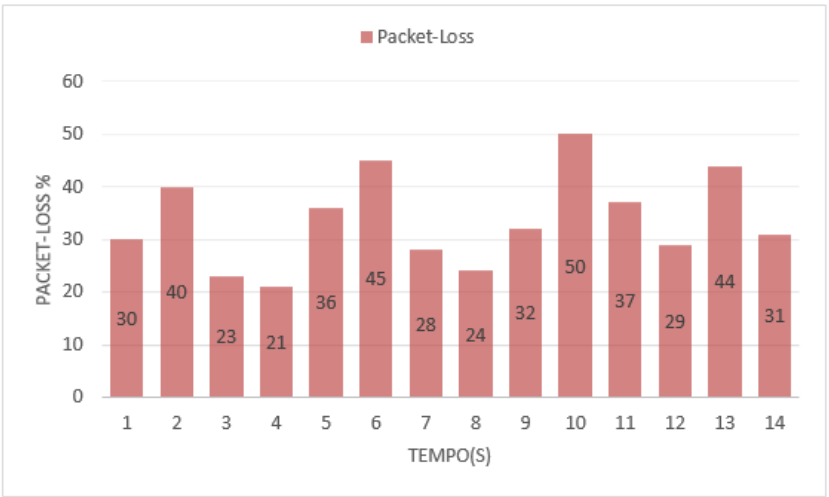


FIG. 6.15: Relatório Iperf para o *Packet-Loss* no Experimento 04 - Cenário C_1

Podemos observar que mesmo utilizando uma banda total igual a 20 Mbps no enlaces entre o *Switch* 1 e o *Switch* 2 os valores de *throughput* são altos e os valores de *packet-loss*

são baixos. Isso mostra que o balanceamento de carga continua sendo favorável na rede SDN.

6.2.6 EXPERIMENTO 05 - BALANCEAMENTO POR APLICAÇÃO (20 MBPS)

No quinto experimento avaliamos a mesma condição do Experimento 03 onde o balanceamento de carga ocorre simulando o envio de duas aplicações distintas levando em consideração que o caminho utilizado tenha uma banda menor, neste caso 20 Mpbs em vez de 30 Mpbs como no Experimento 03. Neste experimento a banda total utilizada também será de 20 Mbits/s. Dos fluxos enviados, foi percebido que o *throughput* da rede variou de 13.1 Mbps a 13.8 Mbps. A maior taxa de transferência foi de 13.8 Mbps nos instantes de 1 e 9 segundos. A Fig. 6.16 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

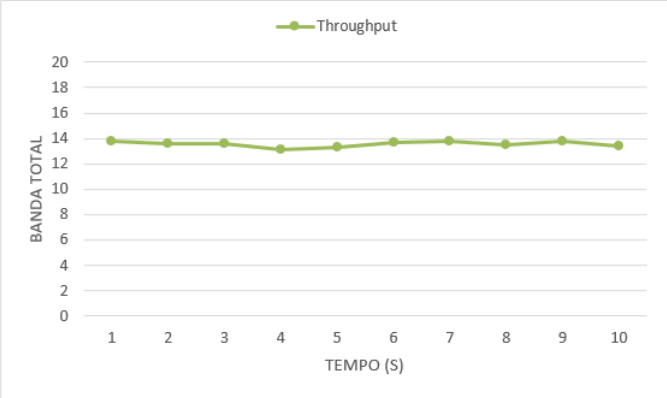


FIG. 6.16: Relatório Iperf para o *Throughput* no Experimento 05 - Cenário C_1

Para os testes de *packet-loss* a quantidade de perda de pacotes é avaliada após o envio dos fluxos. Neste teste foi verificado que o percentual de *packet-loss* variou de 2.3 % a 4.4 %. A maior perda foi de a 4.4 % no instante de 6 segundos. A Fig. 6.17 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do Iperf obtidas neste experimento.

6.2.7 COMPARAÇÃO ENTRE OS EXPERIMENTOS 01, 04 E 05

Após este último experimento realizamos um comparativo dos valores de *throughput* e *packet-loss* obtidos nos Experimentos 01, 04 e 05. Este comparativo tem por finalidade avaliar os benefícios apresentados pela tecnologia SDN em comparação a falta de balance-

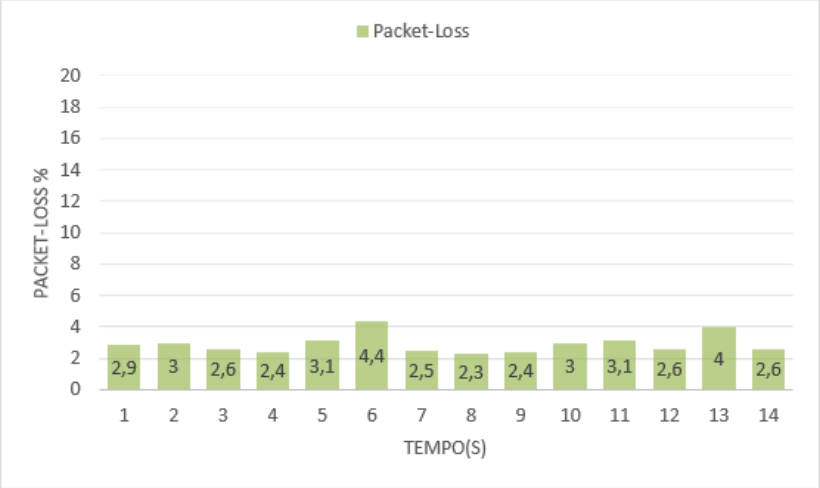


FIG. 6.17: Relatório Iperf para o *Packet-Loss* no Experimento 05 - Cenário C_1

amento de carga. A Fig. 6.19 apresenta um gráfico comparativo sobre os valores obtidos de *throughput* nos Experimentos 01, 04 e 05.

Throughput			
Intervalo	**Experimento 01**	**Experimento 04**	**Experimento 05**
1	2,2	10,2	13,8
2	2,5	10,7	13,6
3	2,1	10,5	13,6
4	2,5	10,6	13,1
5	2,6	10,8	13,3
6	2,8	10,7	13,7
7	2,8	10,7	13,8
8	2,5	10,4	13,5
9	2,3	10,2	13,8
10	2,2	10,8	13,4

FIG. 6.18: Valores *Throughput* Experimentos 01, 04 e 05 - Cenário C_1

Podemos observar que os índices de *throughput* nos Experimentos 04 e 05, os quais utilizam o balanceamento de fluxos LBX ainda são mais significativos aos do Experimento 01 o qual não possui balanceamento de carga, mesmo utilizando a mesma banda total em todos os experimentos. A Fig. 6.20 organiza e apresenta um comparativo dos valores de *throughput* nos Experimentos 01, 04 e 05 em cada instante de tempo.

A Fig. 6.21 apresenta um comparativo sobre o *packet-loss* entre os Experimentos 01, 04 e 05. Analisamos que a porcentagem de *packet-loss* nos Experimentos 04 e 05, os quais utilizam o balanceamento de fluxo ainda são significantemente menores do que os com-

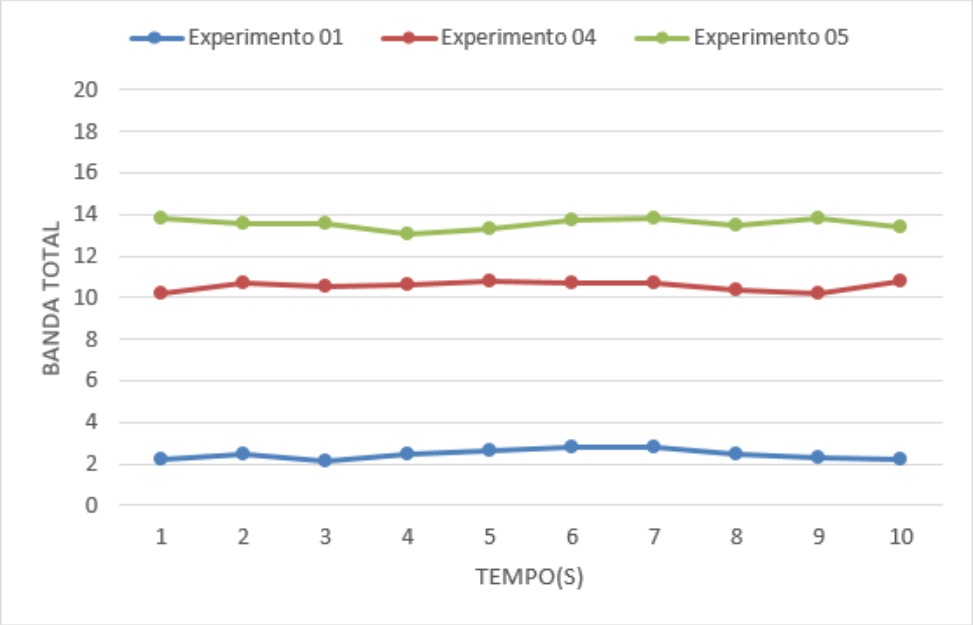


FIG. 6.19: Gráfico Comparativo *Throughput* Experimentos 01, 04 e 05 - Cenário C_1

parados com o Experimento 01, mesmo utilizando a mesma banda total nos experimentos.

Packet-Loss			
Intervalo	**Experimento 01**	**Experimento 04**	**Experimento 05**
1	30	3	2,9
2	40	3,4	3
3	23	2,8	2,6
4	21	2,6	2,4
5	36	3,2	3,1
6	45	4,6	4,4
7	28	2,9	2,5
8	24	2,6	2,3
9	32	2,9	2,4
10	50	3,6	3
11	37	3,5	3,1
12	29	2,9	2,6
13	44	4,3	4
14	31	3	2,6

FIG. 6.20: Valores *Packet-Loss* Experimentos 01, 04 e 05 - Cenário C_1

Os resultados apresentados no cenário C_1 mostraram que a necessidade do balanceamento de fluxos é um problema real nas redes SDN. Notamos que nos experimentos sem o balanceamento dos fluxos foram registrados pelo servidor Iperf inúmeras perdas de pacotes e que o *throughput* da rede manteve-se a baixo comparado ao Experimento 02,

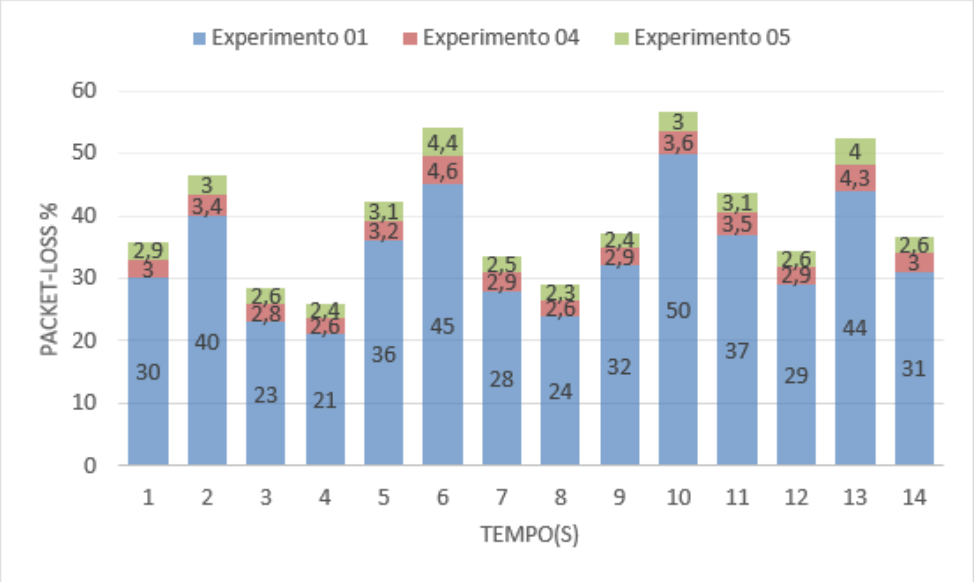


FIG. 6.21: Gráfico Comparativo *Packet-Loss* Experimentos 01, 04 e 05 - Cenário C_1

utilizando o balanceador de fluxos. Com o congestionamento do enlace entre o *Switch* 1 e o *Switch* 2 notamos que o enlace entre o *Switch* 3 fica ocioso e é subutilizado no Experimento 01. Já no Experimento 02 com a aplicação do balanceamento dos fluxos através da aplicação LBX observamos melhorias nos relatórios de *packet-loss* e no *throughput* da rede. Com isso, vemos as vantagens de uma visão centralizada nos testes comparativos e outros benefícios da tecnologia. Observamos que nos experimentos com a aplicação do balanceamento de fluxos todos os enlaces da topologia são utilizados, com isso obtemos um melhor aproveitamento da rede. Os fluxos foram separados de acordo com o valor do identificador obtido através do campo ToS do cabeçalho do protocolo OpenFlow. Com isso foi possível visualizar o armazenamento dos caminhos escolhidos no banco de dados da aplicação. O cenário C_1 de validação foi propício para a geração desses resultados iniciais, porém tornam-se necessárias outras validações em uma topologia com mais nós e com mais possibilidades de escolhas de caminho dentro da rede. A fim de avaliarmos outras perspectivas poderemos entender melhor os objetivos apresentados pela tecnologia SDN em comparação com os modelos obtidos nas redes atuais.

6.3 ANÁLISE DO CENÁRIO C_2

No Cenário C_2 foram realizados 3 experimentos distintos a fim de apresentar também os valores de *throughput* e *packet-loss* em uma rede SDN. Nos experimentos deste cenário

também foram enviados 15 fluxos UDP de 1 Mbps e 30 fluxos TCP de 64 bits cada, entre a origem o Switch 1 e destino o Switch 6. Os fluxos TCP e UDP foram enviados aleatoriamente.

6.3.1 EXPERIMENTO 01 - SEM BALANCEAMENTO DE FLUXOS (10 MBPS)

Neste experimento avaliamos a topologia proposta no Cenário 2 sem a realização do balanceamento de carga. A banda total disponível para o envio dos fluxos é de 10 Mbps. O experimento teve uma duração de 10 segundos. Dos fluxos enviados, foi percebido o *throughput* da rede muito baixo variando entre de 1.1 Mbps a 1.8 Mbps. O maior consumo identificado foi o de 1.8 Mbps no instante de 6 segundos. A Fig. 6.22 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

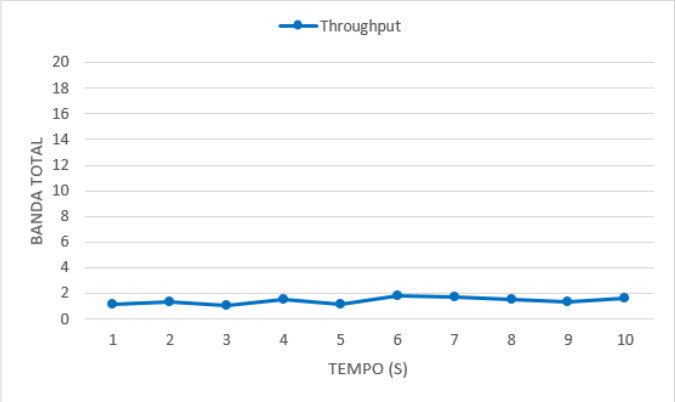


FIG. 6.22: Relatório Iperf para o *Throughput* no Experimento 01 - Cenário C_2

Os testes de *packet-loss* mostraram a quantidade de pacotes perdidos no envio dos fluxos entre a origem e o destino neste cenário. A quantidade de perda de pacotes foi avaliada após o envio os fluxos. Verificamos um percentual muito alto de *packet-loss* neste experimento, o qual variou em torno de 41 % a 88 %. A maior perda foi de a 88 % no tempo de 7 segundos. Novamente a falta de balanceamento de carga influenciou na alta perda de pacotes, tendo em vista que a rede apresenta nós ociosos que poderiam estar sendo utilizados para o balanceamento dos fluxos na rede. A Fig. 6.23 apresenta os valores de *packet-loss* sem a realização do balanceamento de carga.

Assim como no Experimento 01 do cenário C_1, neste cenário foi possível também reproduzir a necessidade do balanceamento de fluxos em uma rede SDN. O baixo *throughput*

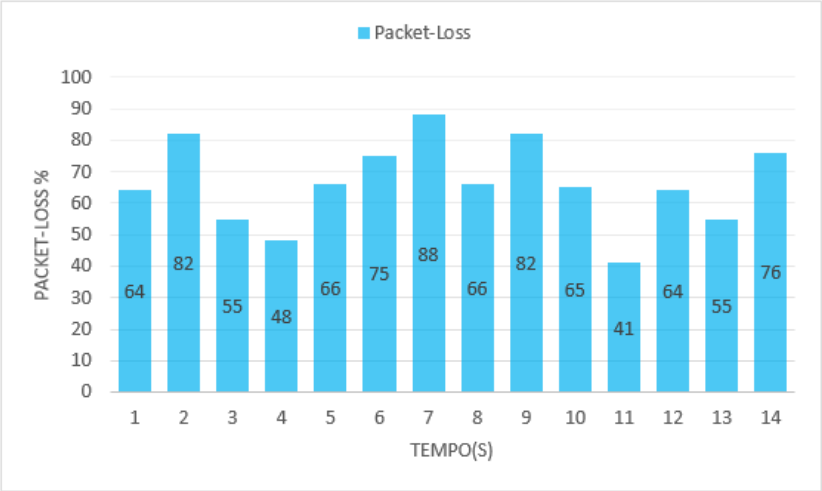


FIG. 6.23: Relatório Iperf para o *Packet-Loss* no Experimento 01 - Cenário C_2

da rede aliado a alta perda de pacotes nos mostra um cenário onde é necessário implementar o balanceamento de carga, para que assim possamos balancear o tráfego de maneira mais justa paras as aplicações trafegadas na rede. Este cenário tambpem mostra-se ideal para a aplicação do balanceador de fluxos LBX.

6.3.2 EXPERIMENTO 02 - COM BALANCEAMENTO DE FLUXOS LBX (30 MBPS)

Neste experimento avaliaremos a topologia proposta com a realização do balanceamento de carga implementada pela aplicação LBX. Os testes foram realizados levando em consideração a distribuição dos fluxos pelos caminhos possíveis. Ao recebimento desses novos fluxos, o *Switch* 1 encaminhará os cabeçalhos desses fluxos para o controlador, que acionará o algoritmo de balanceamento de fluxos LBX. O algoritmo verificará se já existe algum caminho de custo mínimo previamente calculado no banco de dados da aplicação para determinada origem, destino e identificador do fluxo. Caso exista, o algoritmo verificará a possibilidade de existência de um novo caminho disjunto de custo mínimo na rede. Neste experimento foram enviados valores de ToS aleatórios para simularmos fluxos distintos e que os mesmo possam seguir caminhos diferentes na topologia realizando assim o balanceamento dos fluxos. A banda total disponível para o envio dos fluxos é de 30 Mbps. Este experimento teve uma duração de 10 segundos. Dos fluxos enviados, foi percebido o *throughput* da rede variando entre de 12 Mbps a 12.8 Mbps. A maior taxa de transferência foi de 12.9 Mbps no instante de 7 segundos. A Fig. 6.24 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste teste.

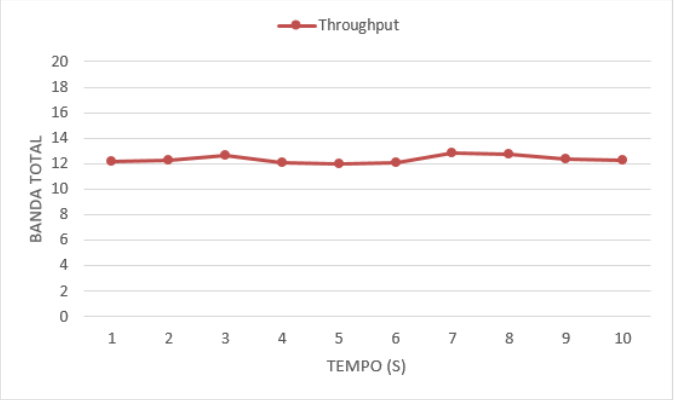


FIG. 6.24: Relatório Iperf para o *Throughput* no Experimento 02 - Cenário C_2

Nesta topologia o algoritmo LBX optou por três possibilidades de caminhos distintos para enviar os fluxos da origem *Switch* 1 para o *Switch* 6. Os caminhos escolhidos são armazenados no Mysql com a finalidade de serem monitorados para escolha de novos caminhos disjuntos pertencentes a mesma origem, destino e identificador de fluxo. O identificador de fluxo é definido a partir do valor do campo ToS do pacote OpenFlow, o qual foi apresentado no Capítulo 4 deste trabalho. Desta maneira podemos separar os fluxos dentro da rede SDN, pois os mesmos são identificados no cabeçalho do pacote OpenFlow. Além do caminho escolhido são armazenados outras informações referentes ao pacote OpenFlow como: DPID Origem, DPID Destino, Endereço MAC Origem, Endereço MAC Destino, Fluxo ID e Caminho escolhido. Cada novo caminho de custo mínimo identificado pelo algoritmo é armazenado no Mysql para que este valor possa ser consultado em uma próxima iteração do algoritmo. A Fig. 6.25 apresenta os registros contidos na tabela lbx do Banco de Dados após a realização deste experimento. Podemos observar que os fluxos que possuem o mesmo identificador são sempre trafegados pelo mesmo caminho na rede SDN.

Os testes de *packet-loss* avaliaram a quantidade de pacotes perdidos no envio dos fluxos entre a origem e o destino baseados nos identificadores dos fluxos da topologia do cenário C_2 . Neste teste foi verificado que o percentual de *packet-loss* variou de 4 % a 5,8 %. A maior perda foi de a 5,8 % no instante de 2 segundos. A Fig. 6.26 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do Iperf obtidas neste experimento.

Neste experimento é observado que o *throughput* é melhor do comparado ao Experimento 01 sem balanceamento de carga. Assim como verificado nos testes da topologia de validação do Cenário C_1, o balanceamento do tráfego continua sendo uma boa es-

SwSrcID	SwDstID	SrcMAC	DstMAC	FlowID	Path
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	1	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	2	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	30	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	1	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	4	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	0	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	18	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	10	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	22	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	38	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	2	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	34	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	40	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	14	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	40	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	20	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	28	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	56	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	8	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	2	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	12	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	1	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	26	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	36	1,4,5,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	44	1,2,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	46	1,3,6
1	6	00:00:00:00:00:00:00:01	00:00:00:00:00:00:00:06	48	1,4,5,6

FIG. 6.25: Relatório da Tabela lbx - Cenário C_2

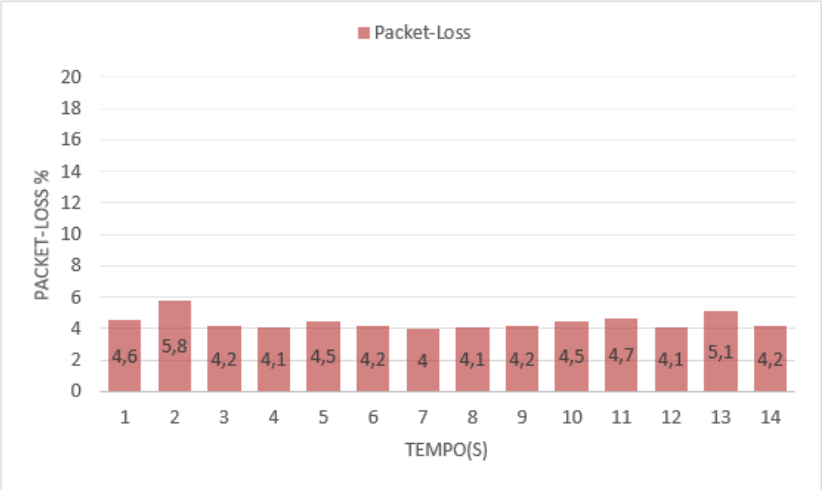


FIG. 6.26: Relatório Iperf para o *Packet-Loss* no Experimento 02 - Cenário C_2

colha, pois ocorre um aproveitamento melhor da banda disponível, pois os fluxos estão seguindo caminhos distintos. É observado também que o *packet-loss* é bastante menor neste experimento.

6.3.3 EXPERIMENTO 03 - ECMP (30 MPBS)

O terceiro e último experimento deste cenário tem como objetivo avaliar o balanceamento de carga através da utilização da solução ECMP, apresentado no Capítulo 2 deste trabalho, o qual é implementado pela empresa Cisco em seus equipamentos de rede.

A solução ECMP é implementada através do modelo DH (Direct Hashing), comentado no Capítulo 2 deste trabalho. O DH realiza o balanceamento da carga baseado na realização de uma função *hash* para as rotas (*Equal Cost Multipath Routing*). A fim de obter um caminho, ele executa um algoritmo de *hash* módulo-K que leva em consideração o identificador do pacote X, obtido através das informações do cabeçalho do pacote. Assim ele aplica uma função *hash* $h(X)$ e extrai o módulo do número dos múltiplos caminhos $mod(h(X))$.

A Cisco implementa o ECMP desta maneira em seus equipamentos de redes. Neste experimento implementamos o ECMP na topologia do cenário C_2. A banda total disponível para o envio dos fluxos é de 30 Mbps. O experimento teve uma duração de em torno de 10 segundos. Dos fluxos enviados, foi percebido que o *throughput* variou de 11 Mbps a 11.8 Mbps para os fluxos TCP. A Fig. 6.27 apresenta as informações sobre o *throughput* no Relatório do Servidor do Iperf obtidas neste experimento.

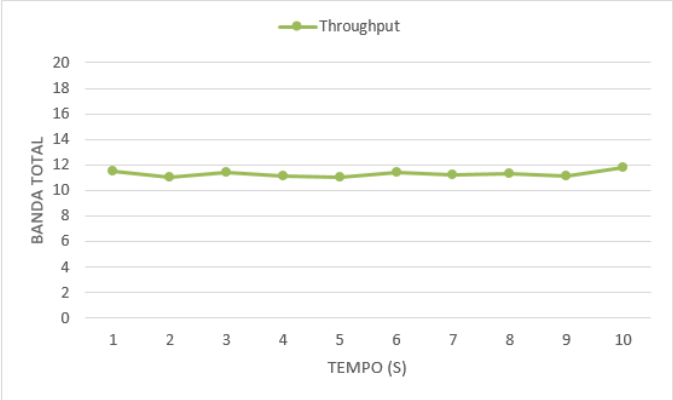


FIG. 6.27: Relatório Iperf para o *throughput* no Experimento 03 - Cenário C_2

Os testes de *packet-loss* os valores variaram de 5 % a 6.8 %. A maior perda foi de a 6.8 % no instante de 1 segundo. A Fig. 6.28 apresenta as informações sobre o *packet-loss* no Relatório do Servidor do Iperf obtidas neste experimento.

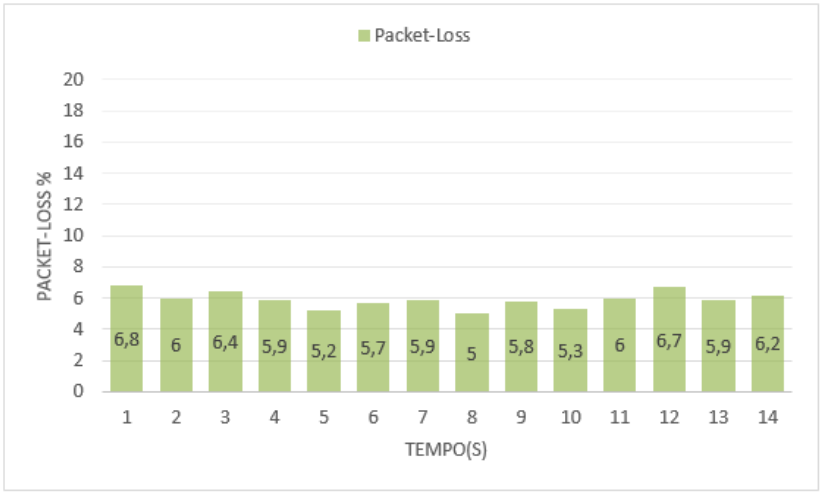


FIG. 6.28: Relatório Iperf para o *Packet-Loss* no Experimento 03 - Cenário C_2

6.3.4 COMPARAÇÃO ENTRE OS EXPERIMENTOS 01, 02 E 03

A Fig. 6.29 apresenta um comparativo sobre as métricas do *throughput* entre os Experimentos 01, 02 e 03 do Cenário C_2. Observamos que os valores do *throughput* e *packet-loss* do Experimento 03 são melhores do que os apresentados pelo Experimento 01, porém são bem similares ao Experimento 02, o qual utiliza o algoritmo LBX. Também podemos comparar as métricas de *throughput* e *packet-loss* obtidas neste experimento através da Fig. 6.31.

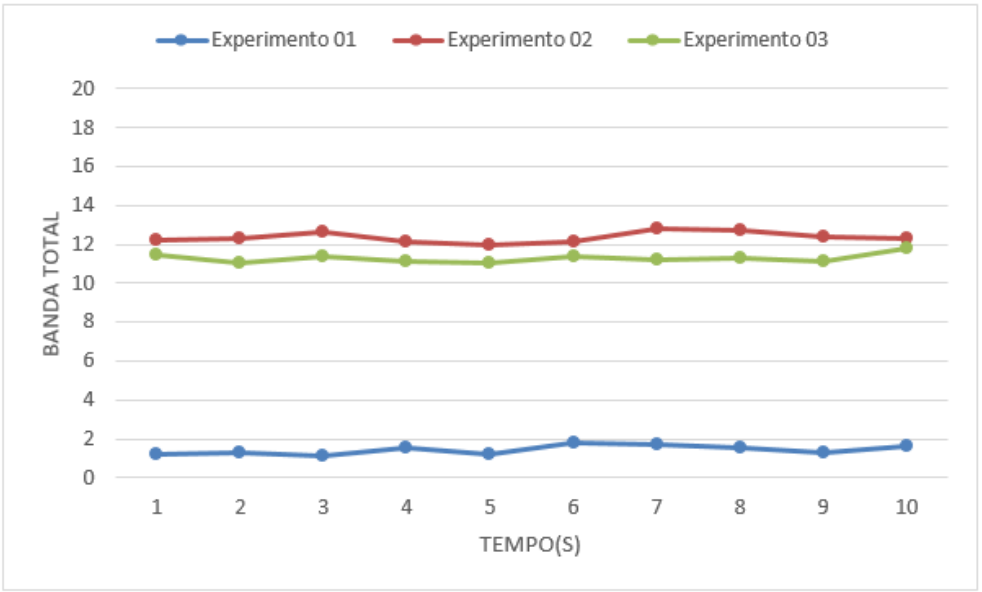


FIG. 6.29: Gráfico Comparativo *Throughput* Experimentos 01, 02 e 03 - Cenário C_2

Throughput			
Intervalo	Experimento 01	Experimento 02	Experimento 03
1	1,2	12,2	11,5
2	1,3	12,3	11
3	1,1	12,6	11,4
4	1,5	12,1	11,1
5	1,2	12	11
6	1,8	12,1	11,4
7	1,7	12,8	11,2
8	1,5	12,7	11,3
9	1,3	12,4	11,1
10	1,6	12,3	11,8

FIG. 6.30: Valores *Throughput* Experimentos 01, 02 e 03 - Cenário C_2

Packet-Loss			
Intervalo	Experimento 01	Experimento 02	Experimento 03
1	64	4,6	6,8
2	82	5,8	6
3	55	4,2	6,4
4	48	4,1	5,9
5	66	4,5	5,2
6	75	4,2	5,7
7	88	4	5,9
8	66	4,1	5
9	82	4,2	5,8
10	65	4,5	5,3
11	41	4,7	6
12	64	4,1	6,7
13	55	5,1	5,9
14	76	4,2	6,2

FIG. 6.31: Valores *Packet-Loss* Experimentos 01, 02 e 03 - Cenário C_2

A aplicação LBX mostrou-se estável para a realização do balanceamento de fluxos pois ela não realiza o *hash* do cabeçalho dos fluxos, como a solução do ECMP. Ele os encaminha baseado nas informações de origem e destino armazenadas no Banco de Dados da aplicação LBX. A solução ECMP implementado pela Cisco distribui os pacotes pertencentes a um determinado fluxo por N caminhos distintos, o que implica na necessidade de reordenação desses pacotes na chegada ao destino desse fluxo. Neste caso torna-se necessário a implementação de um método para o controle da ordenação desses pacotes,

já que eles podem ser enviados fora de ordem. O balanceador de fluxos LBX, avaliado no Experimento 02 deste cenário, não funciona desta forma. Ele seleciona o caminho de custo mínimo baseado no algoritmo de Dijkstra utilizando informações previamente armazenadas no Banco de Dados da aplicação e distribui os fluxos por caminhos disjuntos não sendo necessário a reordenação dos pacotes no seu destino. Com isso, temos vantagens na rede SDN pois não se torna necessária nenhuma reordenação dos pacotes no seu destino. O ECMP implementado pela Cisco, não foi projetado para a tecnologia SDN, pois durante esse experimento foi observado que todos os *switches* trocaram mensagens do tipo *packet-in* com o controlador SDN, pois a cada salto do caminho era necessário recalcular um novo caminho para a entrega do fluxo ao seu destino. Na *API* LBX a partir do momento que o algoritmo seleciona o caminho de custo mínimo e o armazena no Banco de Dados da aplicação, todos os *switches* pertencentes ao caminho tem a sua tabela de roteamento atualizadas ao mesmo tempo para que assim não seja necessária uma outra consulta ao controlador ao longo do caminho referente a determinada origem, destino e identificador de fluxo. Esta é uma das vantagens que a tecnologia SDN nos fornece em comparação as redes legadas. A partir de uma visão global da rede temos possibilidade de aplicarmos políticas na rede diretamente do controlador SDN. Essas ações são simplesmente replicadas para os *switches*da rede.

Os valores de *packet-loss* do Experimento 01 mostraram-se muito altos em comparação com os Experimentos 02 e 03. A Fig. 6.32 apresenta um comparativo sobre os valores de *packet-loss* obtidos nos Experimentos 01, 02 e 03.

O Experimento 02 obteve valores equivalentes de *packet-loss* em comparação com o Experimento 03. Como já comentado, o algoritmo LBX conta com os benefícios da tecnologia SDN e por isso mostra-se equivalente o ECMP implementado pela Cisco. Porém, temos as vantagens oferecidas pela tecnologia SDN, através de uma visão centralizada da rede nos beneficiando com a possibilidade de seleção de múltiplos caminhos além de termos total controle sobre a rede podendo programá-la conforme a nossa necessidade.

Podemos analisar que os resultados entre os Experimentos 02 e 03 no cenário C_2 são muito simulares, apesar das desvantagens do ECMP. As logs apresentadas pelo servidor Iperf para os valores de *packet-loss* mostram que os resultados apresentados no Experimento 03 são decorrentes de entregas de pacote fora de ordem da solução do ECMP. Pelo fato do balanceador de fluxos lidar com fluxos e não com as unidades menores do pacote pode-se observar uma melhoria de valores apresentados pelo Experimento 02. Do

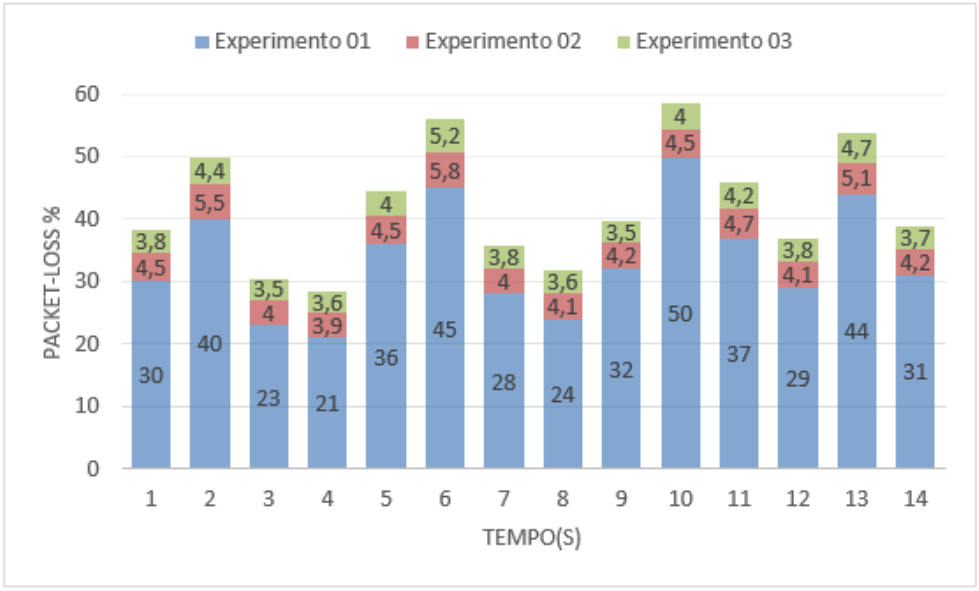


FIG. 6.32: Gráfico Comparativo *Packet-Loss* Experimentos 01, 02 e 03 - Cenário C_2

ponto de vista do balanceamento de carga, a aplicação LBX apresenta vantagens pois lida diretamente com o fluxo das aplicações a partir de uma visão centralizada e não com cada pacote individualmente. Assim podemos entregar uma comunicação fim-a-fim sem a necessidade de reordenação de pacotes na chegada ao seu destino. Visto que através do desempenho e equivalência dos resultados apresentados pela aplicação LBX e pela solução ECMP, notamos que resultado é bom, pois podemos ver que as vantagens oferecidas pela tecnologia SDN são tão válidas quanto as soluções já aplicadas nas redes atuais.

Com os experimentos realizados nos cenários C_1 e C_2 observamos que a aplicação proposta neste trabalho mostrou vantagens com o suporte da tecnologia de redes definidas por *software*. Foi possível a implementação do algoritmo proposto e através dos experimentos foi possível destacar os benefícios dessa tecnologia emergente. Através das logs da tabela lbx e dos resultados do servidor Iperf vimos que o balanceamento de fluxos ocorreu de fato em ambos os cenários abordados. Observamos que o campo ToS, utilizado como campo chave para identificarmos o fluxo dentro da rede SDN, atendeu as expectativas, pois os fluxos divergentes foram trafegados ao longo dos múltiplos caminhos e os fluxos iguais seguiram os mesmo caminho dentro da topologia. O banco de dados da aplicação teve um fator representativo para a execução do algoritmo LBX. Todos os caminhos de custo mínimo selecionados na topologia foram devidamente arquivados e monitorados pela aplicação LBX na busca de novos caminhos disjuntos para o balanceamento dos

fluxos. Vimos que a falta do balanceamento de carga continua sendo um problema atual e também em redes SDN, porém através de SDN temos espaço para inovação no assunto e liberdade de construir novas soluções através de APIs abertas e customizadas. Com esse grande benefício observamos que com SDN podemos programar diretamente a rede, fazendo que a mesma se comporte de acordo com as políticas e regras definidas a partir de uma visão centralizada. Assim podemos melhor definir, modificar e tratar o problema de balanceamento de carga nas redes atuais.

A rede definida por *software* leva a mudanças significativas no modo de criação e de operação das redes de computadores. As redes atuais apresentam diversas demandas que ainda precisam ser discutidas e tratadas. O conceito de redes definidas por *software* está atrelado a capacidade de inovação. Através desta tecnologia, os seus utilizadores podem personalizar e desenvolver novas soluções para as redes de acordo com as suas necessidades e demandas, eliminando ferramentas desnecessárias e criando redes totalmente virtuais e isoladas. SDN é assunto ainda é bastante recente, mas está crescendo em um ritmo muito rápido. Ainda assim, há importantes desafios na pesquisa a serem abordados, tratados e discutidos.

7 CONSIDERAÇÕES FINAIS

7.1 CONCLUSÃO

Neste trabalho foram abordados temas sobre o balanceamento de carga, modelos de roteamento de múltiplos caminhos e redes definidas por *software*. A necessidade de balanceamento de carga nas redes de computadores é um problema real. As redes não foram projetadas para as grandes demandas atuais e por isso veem tentando adaptar-se ao longo dos anos. Os modelos de balanceamento de carga existentes procuram tratar o problema em si mas nenhum deles possui uma visão centralizada da rede e na sua maioria são complexos de se implementar. Os modelos apresentados neste trabalho possuem uma visão parcial da rede e poucos utilizam informações de tráfego e da rede para a escolha do melhor caminho. Topologias com suporte a múltiplos caminhos são cada vez mais reais e a seleção do caminho para o tráfego das informações é cada vez mais urgente. Os equipamentos atuais possuem duas camadas distintas as quais residem no mesmo equipamento. Não há uma separação da lógica da rede e do encaminhamento dos dados. Os equipamentos atuais são proprietários e não oferecem espaço para inovação, pois oferecem soluções proprietárias para problemas específicos. Torna-se complexo integrar redes com equipamentos de rede de fornecedores diferentes, pois cada fornecedor constrói e administra seu hardware e software que conta com bilhares de linhas de código de difícil manutenção. As redes definidas por *software* surgiram com o intuito de modificar a maneira como os administradores de redes gerenciam as redes atuais. Com uma visão mais centralizada da rede e sem a dependência de fornecedores de *hardware* e protocolos proprietários, SDN oferece a possibilidade de programar a rede de acordo com a sua real necessidade, pois temos o plano de dados nitidamente separado do plano de controle que pode ser programado a nível de aplicação. Isso nos trouxe um espaço muito grande para inovação, pois esta tecnologia pode ser facilmente utilizada pra os fins acadêmicos. O projeto de maior destaque para implementação desta abordagem é o protocolo OpenFlow sendo inclusive utilizado pela empresa Google em um dos seus *backbones*.

Neste trabalho foi implementado uma *API* para balanceamento de fluxos, chamada LBX. Esta *API* foi implementada através de um algoritmo de roteamento de custo mínimo,

baseado no algoritmo de Dijkstra, o qual consulta informações sobre os caminhos previamente calculados em um banco de dados da aplicação para a verificação da possibilidade de escolha de um novo caminho disjunto de menor custo. Através do balaceador os fluxos são identificados a partir do valor do campo ToS obtido no cabeçalho do protocolo OpenFlow. O balanceador de fluxos LBX possui uma visão global da rede e foi implementado sem a dependência de qualquer fornecedor de equipamentos de redes. Os experimentos foram realizados utilizando o emulador de redes Mininet e o controlador Floodlight. Utilizamos uma *API* OpenFlow Java aberta para programarmos o algoritmo no controlador Floodlight .

Foram definidos dois cenários distintos (C_1 e C_2) para a validação da *API* LBX. No cenário C_1, o balanceador de fluxos detacou-se com os valores de *troughput* e *packet-loss* apresentados comparado ao experimento sem balanceamento de carga. No Cenário C_2 a o LBX foi comparado com a solução ECMP (*Equal Cost Multipath*) implementado pela empresa Cisco em seus equipamentos. O balanceador LBX obteve resultados equivalentes aos do método ECMP, porém apresentou os benefícios da visão centralizada tecnologia SDN além de espaço de inovação e redução de complexidade através da automação.

Os problemas apresentados demonstraram a necessidade da redução da complexidade das redes atuais, de forma que, a administração possa ser realizada de forma mais centralizada onde os recursos possam deixar de ser proprietários trazendo assim espaço para inovação e melhorias nas redes atuais. Vimos que os protocolos e soluções existentes nas redes atuais são em sua maioria complexas e limitadas pelos seus fabricantes. Dificultando assim o espaço para inovação e para reprodução real desses problemas a fim de se proporem melhorias. As redes atuais tornaram-se "ossificadas" e muito limitadas para atenderem as demandas das aplicações emergentes. Com isso observamos o surgimento de novas tecnologias, com a proposta de uma outra abordagem do problema onde assim poderemos ter uma visão completa da rede podendo assim entender melhor os problemas a fim de propormos novas soluções. Além de fomentar inovação aberta e competição tornado assim a inovação da rede mais fácil e rápida. Toda essa ideia da tecnologia SDN, com a separação das camadas dos equipamentos de redes atuais não surgiu por acaso, ela já vem sendo florescendo ao longo dos anos, através de estudos sobre o quanto a rede tornou-se complexa. SDN já é realidade!

7.2 TRABALHOS FUTUROS

Como trabalhos futuros, pode-se implementar o algoritmo LBX proposto neste trabalho em outros controladores SDN, os quais possam fornecer também suporte as outras versões do protocolo OpenFlow.

O campo ToS do cabeçalho OpenFlow pode ser explorado para a integração com a arquitetura de diferenciação de serviços implementado pelo DiffServ na Qualidade do Serviço (QoS). Assim os fluxos podem ser priorizados dentro da rede SDN, através da aplicação de QoS, além de serem distribuídos ao longo de múltiplos caminhos de custo mínimo.

Outro ponto a ser citado como trabalho futuro é a implementação dos cenários apresentados neste trabalho em uma rede real ou híbrida, com *switches* e roteadores habilitados com o protocolo OpenFlow. A partir desta implementação os resultados obtidos podem ser comparados no ambiente virtual, real e híbrido.

Os dados coletados no banco de dados da aplicação podem ser monitorados por uma aplicação específica de monitoramento dentro da rede SDN, podendo simplesmente apresentar gráficos a respeito da distribuição dos fluxos através dos múltiplos caminhos ou gerar alarmes indicando ociosidade ou sobrecarga de determinados nós na topologia. Por ter uma visão centralizada da rede, o algoritmo LBX pode ser adaptado para coletar informações sobre o status da rede afim de identificar os nós sobrecarregados ou subutilizados na rede SDN e assim poder analisar essa informação também na seleção de caminho na rede. Fazendo com que a escolha de caminhos disjuntos seja mais aprimorada.

E por fim, o algoritmo pode ser adaptado para possuir outros valores de pesos nos enlaces dos *switches* da topologia SDN.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ADISESHU, H., PARULKAR, H. e VARGHESE, G. **A reliable and scalable striping protocol**. *ACM SIGCOMM Computer Communication Review*, 26:131–141, 1996.

AWDUCHE, D., MALCOLM, J., AGOGBUA, J., O'DELL, M. e MCMANUS, J. **requirements for traffic engineering over mpls**, September 1999. http://tools.ietf.org/html/rfc2702.html (Acessado em Abril de 2013).

AWDUCHE, D., NETWORKS, M., CHIU, A., NETWORKS, C., ELWALID, A., WIDJAJA, I. e XIAO, X. **Overview and principles of internet traffic engineering**, May 2002. http://www.ietf.org/rfc/rfc3272.txt (Acessado em Abril de 2014).

CAO, Z., WANG, Z. e SEGURA, E. **Performance of hashing based schemes for internet load balancing**. *IEEE INFOCOM*, págs. 332–341, 2000.

CHEBROLU, K. e RAO, R. **Bandwidth aggregation for real-time applications in heterogeneous wireless networks**. *IEEE Transactions on Mobile Computing*, 5: 388–403, 2006.

CHIM, T., YEUNG, K. e LUI, K. **Traffic distribution over equal-cost-multi-paths**. *Computer Networks*, 49:465–475, 2005.

CONSORTIUM, O. **openflow switch specification v1.3.0**. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (Acessado em Abril de 2014).

FERNANDEZ, J., TALEB, T., GUIZANI, M. e KATO, N. **Bandwidth aggregation-aware dynamic qos negotiation for real-time video streaming in next-generation wireless networks**. *IEEE Transactions on Multimedia*, 11:1082–1093, 2009.

FLOODLIGHT, P. http://www.projectfloodlight.org (Acessado em Abril de 2014).

FOUNDATION, O. N. **the new norm for networks. ofn white paper.**, 2012. https://www.opennetworking.org/sdn-resources/sdnlibrary/whitepapers¿ (Acessado em Abril de 2014).

GROSS, P. **choosing a "common igp" for the ip internet (the iesg's recommendation to the iab)**, October 1992. http://tools.ietf.org/html/rfc1371 (Acessado em Abril de 2014).

HEDRICK, C. e BOSACK, L. **An introduction to IGRP**. *Rutgers-The State University of New Jersey Technical Publication. Laboratory for Computer Science*, 1991.

HOPPS, C. **analysis of an equal-cost multi-path algorithm**, November 2000. https://tools.ietf.org/html/rfc2992 (Acessado em Abril de 2014).

JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ZOLLA, J., HÃ¶LZLE, U., STUART, S. e VAHDAT, A. **B4: experience with a globally-deployed software defined wan**. *ACM SIGCOMM Computer Communication Review*, 43:3–14, 2013.

KANDULA, S., KATABI, D., SINHA, S. e BERGER, A. **Dynamic load balancing without packet reordering**. *ACM SIGCOMM Computer Communication Review*, 37:53–62, 2007.

KIM, J. e AHN, B. **Next-Hop Selection Algorithm over ECMP**. *Communications, 2006. APCC '06. Asia-Pacific Conference on*, págs. 1–5, 2006.

LEE, Y. e CHOI, Y. **An Adaptive Flow-Level Load Control Scheme for Multipath Forwarding**. *Networking-ICN 2001*, págs. 771–779, 2001.

LENGYEL, M., SZTRIK, J. e KIM, C. **Simulation of differentiated services in network simulator**. *Annales Universitatis Scientiarium Budapestinensis de Rolando EÃ¶tvÃ¶s Nominatae, Sectio Computatorica*, 25:85–102, 2005.

LENGYEL, M. e SZTRIK., J. **Performance comparison of traditional schedulers in DiffServ architectures Using NS**. *Proc. the 16th European Simulation Symposium (ESS)*, 2004.

LEUNG, K. e LI, V. **Generalized load sharing for packetswitching networks: Theory and packet-based algorithm**. *IEEE Transactions on Parallel and Distributed Systems*, 17:694–702, 2006.

MALKIN, H. **rfc 2453: Rip version 2**, 1998. https://tools.ietf.org/html/rfc2453 (Acessado em Abril de 2014).

MARTIN, R., MENTH, M. e HEMMEPPLER, M. **Accuracy and dynamics of hash-based load balancing algorithms for multipath internet routing**. *IEEE International Conference on Broadband Communications Networks and Systems (BROADNETS)*, págs. 1–10, 2006.

MCKEOWN, N. e PARULKAR, G. **Software Defined Networks and Openflow, Stanford University**. *SDN CIO Summit 2010*, 2010.

MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S. e TURNER, J. **OpenFlow: enabling innovation in campus networks**. *ACM SIGCOMM Computer Communication Review*, 38:69–74, 2008.

MENDONCA, M., NUNER, B., NGUGYEN, X., OBRACZAKA, K., TURLETTI, T. e AHN, B. **A Survey of Software-Defined Networking: Past, Present and Future of Programmable Networks.** *IEEE Communications Surveys & Tutorials*, págs. 1–18, 2014.

MOY, J. **rfc 2328: Ospf version 2**, 1998. http://www.ietf.org/rfc/rfc2328.txt (Acessado em Abril de 2014).

NETWORKS, J. **network operating system evolution. white paper**, 2010. http://www.juniper.net/us/en/products-services/nos/junos (Acessado em Abril de 2014).

PAREKH, A. e GALLAGER, G. **A generalized processor sharing approach to flow control in integrated services networks: the single-node case**. *IEEE/ACM Transactions on Networking (TON)*, 1:344–357, 1993.

PRABHAVAT, S., NISHIYAMA, H., ANSARI, N. e KATO, N. **Effective Delay-Controlled Load Distribution over Multipath Networks**. *Parallel and Distributed Systems, IEEE Transactions on*, 22:1730–1741, 2011.

REITBLATT, M., FOSTER, N., REXFORD, J. e WALKER, D. ***Consistent updates for software-defined networks: change you can believe in!*** *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 7, 2011.

REKHTER, Y., LI, T. e HARES, S. **a border gateway protocol 4 (bgp-4)**, January 2006. http://tools.ietf.org/html/rfc4271 (Acessado em Abril 2014).

ROSEN, E., VISWANATHAN, A. e CALLON, R. **multiprotocol label switching architecture**, January 2001. http://www.ietf.org/rfc/rfc3031.txt (Acessado em Abril de 2014).

ROTHENBERG, C. **OpenFlow e redes definidas por software: um novo paradigma de controle e inovaÃ§Ã£o em redes de pacotes**. *Cad. CPqD Tecnologia*, 7:65–76, 2010.

ROY, R. e MUKHERJEE, B. **Degraded-Service-Aware Multipath Provisioning in Telecom Mesh Networks**. *Optical Fiber Communication Conference*, págs. 1–3, 2008.

SHI, W., MACGREGOR, M. e GBURZYNSKI, P. **Load balancing for parallel forwarding**. *IEEE/ACM Transactions on Networking*, 13:790–801, 2005.

SIMON, S. **a arpanet**, 1997. http://www.ime.usp.br/ is/abc/abc/node20.html (Acessado em Abril de 2014).

SINGH, R. K., CHAUDHARI, N. S. e SAXENA, K. **Load Balancing in IP/MPLS Networks: A Survey**. *Communications and Network*, 4, 2012.

SIRIPONGWUTIKORN, P., BANERJEE, S. e TIPPER, D. **Traffic Engineering in the Internet: A Survey of Load Balanced Routing**. *Communications of the ACM*, 2002.

SONG, J., KIM, S., LEE, M., LEE, H. e SUDA, T. **Adaptive load distribution over multipath in NEPLS networks**. *Communications, 2003. ICC '03. IEEE International Conference on*, 1:233–237, 2003.

THALER, D. **multipath issues in unicast and multicast next-hop selection**, 2000. http://tools.ietf.org/html/rfc2991 (Acessado em Abril de 2014).

THALER, D. e RAVISHAHKAR, C. **Using name-based mappings to increase hit rates**. *IEEE/ACM Transactions on Networking*, 6:1–14, 1998.

VILLAMIZAR, C. **ospf optimized multipath (ospf-omp)**, 1999. http://tools.ietf.org/html/draft-ietf-ospf-omp-02 (Acessado em Abril de 2014).

WANG, Y. e WANG, Z. **Explicit routing algorithms for internet traffic engineering**. *Computer Communications and Networks, 1999. Proceedings. Eight International Conference on*, págs. 582–588, 1999.

ZININ, A. **Cisco IP routing: packet forwarding and intra-domain routing protocols**. 2002.