UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

BALANCEAMENTO DE CARGA EM SDN UTILIZANDO TABELAS DE GRUPO OPENFLOW

Géssica Fontenele Torquato

Orientador
SIDNEY LUCENA

RIO DE JANEIRO, RJ – BRASIL
05 DE 2019

Catalogação informatizada pelo(a) autor(a)

T687

Torquato, Géssica Fontenele
Balanceamento de carga em SDN utilizando tabelas de grupo OpenFlow / Géssica Fontenele Torquato. -- Rio de Janeiro, 2019.
55 f.

Orientador: Sidney Cunha de Lucena.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2019.

1. SDN. 2. Balanceamento de carga. 3. OpenFlow. I. Lucena, Sidney Cunha de, orient. II. Título.

BALANCEAMENTO DE CARGA EM SDN UTILIZANDO TABELAS DE GRUPO OPENFLOW

Géssica Fontenele Torquato

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

_____

SIDNEY LUCENA (UNIRIO)

_____

MORGANNA CARMEM DINIZ (UNIRIO)

_____

MAXIMILIANO MARTINS DE FARIA (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

05 DE 2019

RESUMO

Este trabalho se propõe a realizar um estudo do paradigma de redes definidas por software, apresentando sua arquitetura e principais conceitos envolvidos, como o protocolo OpenFlow. Sendo o balanceamento de carga umas das primeiras perspectivas de aplicação das redes definidas por software, também apresenta-se algumas da principais estratégias propostas com esse fim. Usando as ferramentas Mininet, o Open vSwitch e o controlador remoto Ryu, um ambiente de testes é criado a fim de mostrar a aplicação dos conceitos OpenFlow de tabelas de grupo e buckets realizando o balanceamento de carga na rede. Como resultado, tem-se que o tráfego foi balanceado de forma equilibrada entre os buckets, como foi observado pela taxa de pacotes transmitidos em cada interface de saída do switch.

Palavras-chave: SDN, Balanceamento de carga, OpenFlow.

ABSTRACT

The present work aims to provide a research on the software defined networking paradigm. Architecture and the key concepts like OpenFlow are shown. As load balancing was one of the first applications conceived for SDN/OpenFlow, some of the different techniques that have been proposed on this theme are mentioned. Using tools like Mininet, Open vSwitch and Ryu controller, a testbed is set to analyze the application and performance of OpenFlow's group tables and buckets in load balancing. As result, it is seen that the packets were fairly balanced among the buckets.

Keywords: SDN, Load Balancing, OpenFlow.

Índice

Índice de Figuras

1 – Introdução

1.1 Motivação

As tecnologias que necessitam de conexão à internet estão inseridas em praticamente todas as atividades diárias da sociedade contemporânea. Estão presentes nas páginas web que viabilizam o comércio eletrônico, nos sistemas de internet banking, nas plataformas de e-mail, na educação à distância, nos aplicativos de navegação baseados em GPS e mais recentemente, em dispositivos domésticos, como smartphones e televisores (IoT). Inúmeras iniciativas de inclusão digital são desenvolvidas por órgãos governamentais e privados, com o objetivo de expandir ainda mais seu alcance para toda a população. Os usuários, as corporações e os serviços públicos estão totalmente dependentes desta infraestrutura de rede. Esta importância gera um problema para o desenvolvimento de novas tecnologias e protocolos de comunicação, pois as pesquisas não podem ser realizadas diretamente no ambiente de produção (aquele disponível e utilizado pelos usuários) devido ao risco de interrupção dos serviços essenciais.

A fim de conquistar mais flexibilidade, tornando mais fácil criar e introduzir novas abstrações na rede, uma abordagem possível é dividir o problema do controle da rede em partes tratáveis: separar o plano de dados do plano de controle. O que leva à definição de redes definidas por software:

> É a separação física do plano de controle da rede do plano de dados. Essa arquitetura desagrega o controle da rede e o encaminhamento, permitindo que o controle de rede se torne diretamente programável e a infraestrutura subjacente seja abstraída pelas aplicações e serviços. (FOUNDATION,2013)

O paradigma de redes definidas por software é cada vez mais pesquisado tanto pela indústria quanto pelas universidades, e tem sido amplamente adotada no mundo corporativo, revolucionando o gerenciamento de redes corporativas de larga escala, infraestruturas com base na nuvem e redes de data centers.

Vários fatores impulsionam os administradores de rede a escolher implementar o conceito de redes definidas por software, que também são usualmente chamadas de SDN (*Software-Defined Networking*). Por exemplo, é consenso que redes tradicionais podem ser

muito complicadas de gerenciar, muito caras e sujeitas à calcificação. Em redes de grande porte, a tentativa de realizar alterações pode envolver configurações de centenas de dispositivos com diversas características e parâmetros de configuração. Além disso, os administradores de rede frequentemente precisam reconfigurar a rede para evitar loops, ganhar velocidade de convergência de rotas e priorizar uma determinada classe de aplicativos.

Esta complexidade no gerenciamento resulta do fato de que cada dispositivo de rede possui a lógica de controle e a lógica de encaminhamento de dados integradas. Por exemplo, em um roteador de rede, protocolos de roteamento como RIP ou OSPF constituem a lógica de controle que determina como um pacote deve ser encaminhado. Os caminhos determinados pelo protocolo de roteamento são codificados em tabelas de roteamento, que são então utilizadas para encaminhar os pacotes. De forma similar, em um dispositivo da camada 2, como um switch, parâmetros de configuração constituem a lógica de controle que determina o caminho dos pacotes. Portanto, o plano de controle de uma rede tradicional é distribuído na estrutura de dispositivos de rede; em consequência, alterar o comportamento de encaminhamento de uma rede envolve mudanças nas configurações de muitos (potencialmente todos) dispositivos de rede. Efetuar alterações na configuração da rede ao escalar para sistemas maiores, ou reduzir em tamanho, pode ser entediante porque um engenheiro de rede precisaria levar em consideração cada dispositivo de rede que pode ser afetado pela mudança. Tradicionalmente, isso tem sido feito por meio de pacotes caros de software de automação ou scripts caseiros que fazem interface com a linha de comando de cada dispositivo de rede.

Com a abstração do plano de controle para um local central, as mudanças na rede consistem em alterações mais simples efetuadas por meio de programação. Estas alterações são inseridas em um controlador centralizado que, por sua vez, atualiza todas as tabelas de encaminhamento em cada dispositivo de rede. Estas alterações podem ser testadas com antecedência, introduzidas, confirmadas e programadas para serem aplicadas em todo o sistema. Como resultado, cada vez mais organizações consideram a implementação de redes definidas por software como a solução para esses problemas.

Atualmente, o paradigma de redes definidas por software já está estabelecido como arquitetura de rede viável para datacenters e *backbones*. Consideremos como exemplo o projeto do Google que foi pioneiro no uso de uma WAN definida por software. Sua rede entre datacenters - batizada de B4 - funciona seguindo uma arquitetura SDN/OpenFlow. A B4 tem

dois backbones: um é chamado I-Scale, voltado para o tráfego cotidiano, que requer alta disponibilidade e sensibilidade à perdas. O outro *backbone* é chamado G-Scale, voltado para o tráfego interno de datacenters, que é volumoso, mas pode tolerar altas taxas de perdas e é menos rigoroso em exigência de disponibilidade . O G-Scale controla a maior parte do tráfego leste-oeste que está crescendo a uma velocidade muito maior do que o tráfego norte-sul tratado pelo I-Scale. A maior vantagem da B4 é proporcionar uma melhor utilização dos links existentes. O serviço de engenharia de tráfego centralizado da B4 impulsiona a utilização dos links para perto de 100%, enquanto a divisão das aplicações flui entre vários caminhos para equilibrar a capacidade a partir da demanda e dos pedidos de prioridade (Ma, 2014). Outras importantes organizações como AT&T, Microsoft e Verizon também adotam SDN para reduzir custos, melhorar a utilização e oferecer novos serviços orientados ao cliente.

Em pesquisa recente, Cox *et al* (2017) apresenta um levantamento sobre o ensino, pesquisa e implementação de redes definidas por software dentre as 100 universidades americanas mais bem colocadas no *ranking* nacional. Para os fins desta pesquisa, o referido autor considera que uma universidade está ativamente pesquisando SDN se, a partir de 2014, houver realizado pelo menos uma publicação relacionada ao tema, lecionado SDN como parte do curso, ou participado de algum projeto correlato. Ao fim deste levantamento, tem-se que aproximadamente 81% destas universidades estão ativamente desenvolvendo SDN como um paradigma de redes viável. O autor também identificou também que 40% destas instituições têm SDN implantado em seus *campi* (mesmo que parcialmente).

Instituições e agências governamentais americanas são mais lentas na adoção das redes definidas por software, mas têm dado passos decisivos neste sentido, apresentando declarações relativas a implantações recentes indicando benefícios tais como redução de custos de equipamentos e melhoria de desempenho da rede. Outros benefícios trazidos pela adoção de SDN nas infraestruturas de redes do governo americano são apontados em uma pesquisa da Juniper: melhor desempenho e eficiência da rede, operações de rede simplificadas, redução de custos nas operações, mais agilidade via automação e orquestração, serviços melhorados e maior segurança. Da mesma forma, ao implantar soluções SDN, as organizações governamentais adotam os seguintes critérios, em ordem de prioridade: alta disponibilidade e resiliência, capacidade de análise e geração de relatórios, automação e provisionamento rápido, opções de código aberto e escalabilidade (COX *et al*,2017).

No âmbito militar, oficiais consideram a SDN como um multiplicador operacional dentro de suas futuras redes. Em 2016, um relatório divulgado pelo *Chief Information Officer* do Exército americano indica que as estratégias de investimento em rede devem incluir SDN. Este mesmo relatório promove a SDN como uma forma de criar redes gerenciáveis e adaptáveis que também são mais ágeis, rentáveis e capazes de serem rapidamente configuradas e reimplantadas conforme necessário (COX *et al*,2017).

O paradigma SDN oferece flexibilidade no controle de grandes redes e eficiência no gerenciamento do tráfego de dados. Simultaneamente, a demanda crescente por serviços baseados em nuvem e por mobilidade são fatores que influenciam no crescimento desse mercado. Relatórios especializados do mercado de redes definidas por software preveem que as receitas excedam a marca de U$100 bilhões anuais até 2020 (SDxCentral,2015).

Dentre os principais *players* do mercado de redes definidas por software destacam-se as empresas americanas Cisco Systems, Hewlett Packard Enterprise, IBM Corporation, Intel Corporation, Juniper Networks, Intersil, VMware, Inc. e a chinesa Huawei Technologies entre outros.

O desenvolvimento de pesquisas na área de redes definidas por software tem grande relevância. Empresas como Google, Facebook, Yahoo, Microsoft, Verizon e Deutsche Telekom fundaram em 2011 a Open Networking Foundation (ONF), fundação cujo principal objetivo é a promoção e adoção da SDN. A ONF é um consórcio sem fins lucrativos, dedicado ao desenvolvimento, padronização e comercialização da SDN.

1.2 Objetivo

Este projeto tem como objetivo apresentar um estudo sobre o paradigma de redes definidas por software, e realizar um estudo de caso mostrando a aplicação de conceitos OpenFlow no balanceamento de carga em SDN. Para isso, uma rede SDN é implementada com o emulador de rede Mininet e o controlador Ryu, onde é executado o algoritmo de balanceamento *Multipath Routing*.

1.3 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo II: apresentamos uma revisão teórica, introduzindo o conceito de SDN, apresentação do modelo de referência para arquitetura SDN sugerido pela ONF -*Open Networking Foundation*. Apresentação de cada camada e das interfaces *southbound* e *northbound*. Apresentamos ainda o protocolo OpenFlow: conceito ,características dos dispositivos que implementam OpenFlow e ações básicas que o OpenFlow pode realizar para cada novo fluxo. O switch OpenFlow : os componentes de um switch OpenFlow: tabelas de fluxos, uma tabela de grupo, uma tabela de medições e um canal seguro; os tipos de tabelas de grupo; os tipos de mensagens geradas pelo switch OpenFlow: mensagens controlador-switch , mensagens assíncronas, mensagens simétricas. O conceito e arquitetura do controlador SDN: discorre sobre as funções e importância do controlador na arquitetura SDN. Descoberta de topologia de rede em SDN.

- Capítulo III: revisamos conceitos relacionados ao balanceamento de carga. As principais estratégias de balanceamento de carga em redes tradicionais: balanceamento de carga por destino, balanceamento de carga por fluxo, balanceamento de carga por pacotes. Estratégias para o balanceamento de carga em redes SDN: breve apresentação de várias propostas relevantes publicadas no âmbito acadêmico. Apresentação do algoritmo utilizado no estudo de caso.

- Capítulo IV: apresentamos o ambiente de testes, a topologia e as ferramentas Mininet, Ryu, Open Vswitch e o Iperf. Também mostramos os resultados da implementação da rede e balanceamento do tráfego.

- Capítulo V: apresentamos as considerações finais, as contribuições da pesquisa e possibilidades de aprofundamento posterior.

2 – Revisão teórica

2.1- Redes definidas por software - SDN (*Software-Defined Networking*)

O paradigma das redes definidas por software separa o plano de controle (a lógica de controle da rede) do plano de dados (switches e roteadores que encaminham o tráfego). Com a desagregação dos planos de controle e de dados, os switches se tornam simples dispositivos de encaminhamento e a lógica de controle é implementada em um controlador centralizado, simplificando a reconfiguração e avaliação da rede.

As redes tradicionais são complexas e difíceis de gerenciar. Para realizar uma configuração desejada, os administradores precisam configurar individualmente cada dispositivo de rede, usando comandos de baixo nível, muitas vezes específicos para cada fornecedor. Além da complexidade de configuração, ambientes de rede precisam ser capazes de tolerar eventuais falhas e se adaptar às mudanças de carga nos enlaces.

> Uma enorme vantagem trazida pelas redes definidas por software consiste na capacidade de controle que os administradores do sistema têm durante a ocorrência de mudanças. Não haverá mais interrupções da rede cada vez que um cabo for conectado ou desconectado (por exemplo, quando um técnico conectar acidentalmente um cabo a uma porta não utilizada entre switches); as interrupções só acontecerão quando as alterações de configuração forem submetidas ao controlador. As portas não utilizadas são desativadas porque a rede não está programada para enviar quaisquer pacotes da porta e quaisquer novas tentativas de comunicação que aparecerem naquela porta têm que ser permitidas pelo controlador ou pré-programadas para a porta. Com a SDN, somente serão impactados os caminhos nos quais houver falha do link ou alterações (BOBBA,2014).

Uma definição mais detalhada de SDN é encontrada em Kreutz et al (2015), que explica tratar-se de uma arquitetura de rede baseada em quatro pilares:

1) a supracitada separação entre os planos;

2) as decisões de encaminhamento são baseadas em fluxos. Um fluxo é definido por um conjunto de valores no cabeçalho do pacote que funciona como um critério para filtrar e um conjunto de ações (instruções). No contexto SDN, um fluxo é uma sequência de pacotes entre a origem e o destino. Todos os pacotes de um fluxo recebem as mesmas políticas de

serviços. A abstração em fluxos permite unificar o comportamento de diferentes tipos de dispositivos de rede. A programação de fluxos permite uma grande flexibilidade, limitada apenas pela capacidade das tabelas de fluxos;

3) o controle lógico é movido para uma entidade externa, o controlador SDN também chamado de *Network Operating System* (NOS). O controlador é uma plataforma que roda em um servidor *commodity* e oferece facilidade para programar dispositivos de encaminhamento baseados em uma lógica centralizada, abstraída da visão da rede;

4) a rede é programável através de softwares rodando no controlador e interagindo com os dispositivos físicos.

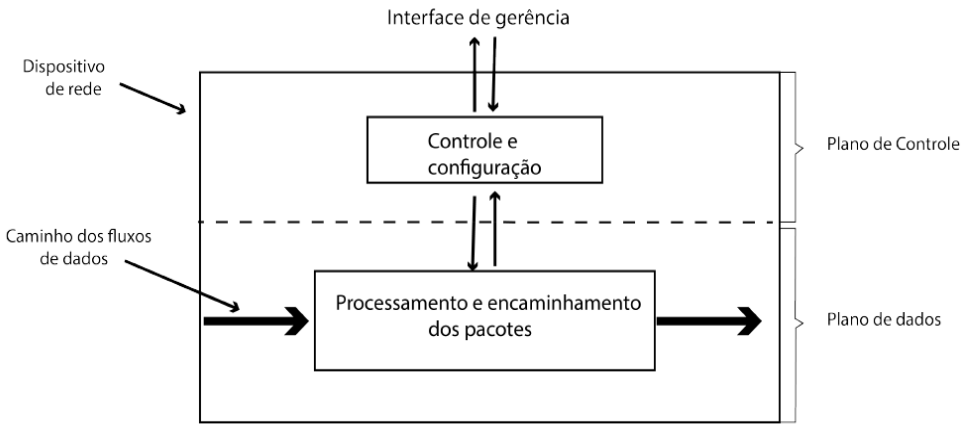

Figura 1: Plano de controle e de dados
Fonte: Comer *apud* Kleis, 2014, p.18.

A separação entre o plano de controle e o plano de dados (Figura 1) pode ser realizada por meio de uma API (*Application Programming Interface*). O exemplo mais famoso é o OpenFlow. Embora SDN e OpenFlow tenham começado como pesquisas no âmbito acadêmico, ganharam importância na indústria. Grande parte dos fornecedores de switches já incluem suporte ao OpenFlow.

Por definição, as redes definidas por software baseiam-se na separação dos planos de dados e de controle, mas também são fundamentadas na programabilidade do plano de controle. No entanto, nenhuma dessas características é totalmente nova em uma arquitetura de rede. Várias abordagens anteriores buscaram promover a programabilidade da rede. Um exemplo é o conceito de rede ativa (*active networking*), que busca controlar a rede em tempo real através de software. O SwitchWare é uma solução de *active networking*, que permite que os pacotes através da rede possam modificar operações da rede dinamicamente. Similarmente,

suítes como Click, XORP, Quagga e BIRD, também são tentativas de tornar dispositivos de rede programáveis. O comportamento desses dispositivos pode ser modificado através do software de roteamento.

Existem soluções centradas no fornecedor para configuração e monitoramento dos switches de rede de toda a empresa. Um exemplo é o produto Cisco® Network Assistant. Estas ferramentas atendem parcialmente às necessidades do administrador de rede para visualizar e configurar os equipamentos de rede. Dependendo das capacidades do software e dos switches de rede que estão sendo gerenciados, um administrador pode projetar uma topologia de rede, monitorar a utilização de recursos, habilitar ou desabilitar portas de switches, ativar uma configuração salva para um switch, ativar uma atualização de firmware, ou fazer o backup da configuração de um switch. Embora todos esses recursos pareçam ótimos, eles não contêm um conjunto completo de funções. Por exemplo, o software do fabricante opera tipicamente apenas com equipamentos de switch de rede do mesmo fornecedor. Outra deficiência é a necessidade de configurar individualmente os switches; não há nenhuma configuração padrão para todos os switches. Com as redes definidas por software, as diferenças entre fornecedores são unificadas sob o conjunto de regras da tabela de encaminhamento regidas pelo protocolo de comunicação entre o dispositivo de rede e o controlador de fluxo. Por exemplo, se múltiplos fornecedores suportarem o protocolo OpenFlow 1.3, os vários produtos dos fornecedores podem ser todos programados pelo mesmo controlador. O operador que estiver inserindo as configurações da rede não precisa saber se os dispositivos de rede do campo são todos do mesmo fornecedor ou de vários fornecedores (BOBBA,2014).

As redes definidas por software evoluíram a partir de trabalho realizado na Universidade de Berkeley e Stanford no ano de 2008, mas a ideia de separar os planos de dados e de controle já era bastante trabalhada desde a última década. Em 2004, foi apresentado o *Routing Control Platform* (RCP), que substituía o roteamento inter-domínio *Border Gateway Platform* (BGP) por um controle de roteamento centralizado para reduzir a complexidade de computação do caminho completamente distribuído. Nesse mesmo ano, IETF lançou o *framework* ForCES, que separava controle e encaminhamento de pacotes em uma rede ForCES. Em 2006, foi apresentada a arquitetura *Path Computation Element* (PCE), que computava caminhos separadamente do encaminhamento de pacotes em redes MPLS e GMPLS. Em 2007, foi apresentado o paradigma de redes Ethane, em que um controlador centralizado gerenciava a admissão e roteamento de fluxos. Nesse último caso, o princípio de

separação de plano de dados e de controle foi explicitamente estabelecido. Dispositivos de rede comerciais também adotaram a ideia de separação dos planos de dados e controle. Por exemplo, em roteadores Cisco ASR 1000 e switches Nexus 7000, o plano de controle é separado do plano de dados e modularizado (WENFENG et alli,2015).

As principais vantagens do SDN derivam do fato de que a camada de controle é separado do encaminhamento, por isso temos que a rede é diretamente programável, tem mais agilidade, por permitir aos administradores ajustar dinamicamente o fluxo de tráfego pela rede conforme necessário. Outro ponto forte é que a inteligência da rede é logicamente centralizada em softwares controladores que mantém uma visão global da rede.

> Com redes definidas por software, (...) todo esse modelamento do tráfego poderá ser feito através de um console de gerência, e em instantes, os dispositivos da rede seriam configurados, não havendo mais a necessidade de conectar-se a cada dispositivo. Um exemplo similar pode ser visto no webinar da Cisco (...) , onde, num primeiro momento, um *streaming* de vídeo não era executado corretamente devido a baixa prioridade e saturação do link principal. Após rápidos ajustes no controlador, o link secundário (redundância) que antes estava ocioso, passou a ser utilizado e o *streaming* foi feito corretamente (KLEIS,2015,p 21).

2.1.1 – Arquitetura SDN

A ONF sugere um modelo de referência para arquitetura SDN, que consiste de três camadas, como mostra a Figura 2:

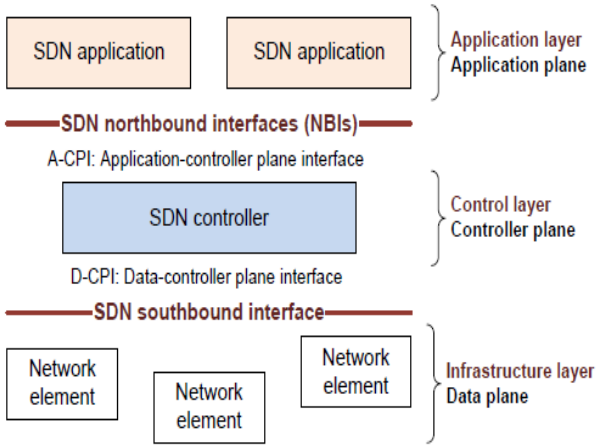

Figura 2 : Arquitetura de redes definidas por software

Fonte: Foundation, 2014, p.13.

- A camada da infraestrutura (plano de dados): consiste de switches e roteadores. Esses dispositivos quase sempre tem dupla função, são responsáveis por coletar status da rede (dados como topologia da rede, estatísticas de tráfego e uso da rede), armazenar esses dados temporariamente em memória local e então enviá-los ao controlador. Também são encarregados do processamento dos pacotes, obedecendo às regras definidas pelo controlador.

As interfaces southbound são pontes de conexão entre a camada de infraestrutura e a camada de controle e dispositivos de encaminhamento. Essa interface, que também é um API, define o conjunto de instruções dos dispositivos de encaminhamento, e também define o protocolo de comunicação entre eles e os elementos do plano de controle. A interface southbound mais usada é a OpenFlow, que será descrito adiante.

- A camada de controle (plano de controle): faz uma ponte entre a camada de infraestrutura e a camada de aplicação. Essa camada é constituída basicamente pelo controlador, que funciona como uma inteligência logicamente centralizada que possui uma visão global da rede e tem capacidade de reconfigurá-la dinamicamente (WENFENG, 2015).

Para interagir com a camada de aplicação, a camada de controle usa a interface *northbound*, que é uma API que abstrai instruções de baixo nível usadas pelas interfaces southbound para programar dispositivos de encaminhamento. A função principal de uma interface *northbound* é traduzir os requisitos das aplicações de gerenciamento em instruções de baixo nível para os dispositivos da rede e transmitir estatísticas sobre a rede, que foram geradas nos dispositivos da rede e processadas pelo controlador.

- A camada de aplicação (plano de aplicação): contém as aplicações SDN que atendem às necessidades dos usuários, que no caso são os administradores da rede. Através de uma plataforma programável fornecida pela camada de controle, as aplicações podem acessar e controlar switches na camada de infraestrutura. Como exemplo de funções dessas aplicações, destacam-se: roteamento, balanceamento de carga e aplicação de políticas de segurança. Além dessas, algumas funções mais novas são realizadas, como economia de energia, aplicação de Qualidade de Serviço (*Quality of Service* – QoS) fim-a-fim, virtualização de redes, gerenciamento de mobilidade em redes sem fio, engenharia de tráfego, e muitas outras.

Para que o funcionamento de uma SDN seja possível, é necessário que exista um protocolo de comunicação entre os diferentes elementos da rede e o controlador, de forma que este último seja capaz de manipular as tabelas de fluxos dos componentes da rede. O protocolo mais famoso e com estágio de desenvolvimento mais avançado é o projeto *open source* OpenFlow, que proporciona uma interface aberta padrão para que a comunicação seja possível.

2.2 - Protocolo OpenFlow

O protocolo OpenFlow é um padrão de código aberto, que foi proposto para uniformizar a comunicação entre os switches e o controlador na arquitetura SDN. Portanto, desempenha papel fundamental nesta arquitetura. Ao atuar como uma interface *southbound* entre as camadas de controle e dados, ele implementa as regras de encaminhamento que tornam o conceito de SDN possível.

Um dispositivo de rede que implementa OpenFlow, tem que satisfazer três características fundamentais:

1- ter uma tabela de fluxos e, para cada fluxo, uma ação a tomar: OpenFlow se utiliza principalmente das tabelas de fluxo, presentes na maioria dos switches e roteadores modernos. Cada switch OpenFlow tem sua própria tabela, que por analogia corresponderia às tabelas MAC e tabela de roteamento nos switches tradicionais. A tabela de fluxos dentro do comutador OpenFlow identifica os fluxos para que o plano de dados execute ações aplicáveis aos pacotes que são pertencentes àquele fluxo. Um fluxo é a representação de um ou mais pacotes em função de suas características. Um fluxo pode incluir pacotes que ainda não passaram pelo switch mas que possuem as mesmas características que compõem aquele fluxo. Assim, a ação associada ao fluxo é aplicada a esses novos pacotes. (PANTUZA, 2016);

2 - um canal seguro, que será utilizado para toda a comunicação entre os elementos de rede e o controlador. O canal seguro é a interface que conecta cada switch OpenFlow ao controlador. Através dessa interface, o controlador configura e gerencia o switch, recebe mensagens de eventos do switch e envia pacotes;

3- utilizar o protocolo OpenFlow para toda a comunicação previamente mencionada, o que proporcionará uma interface padrão que permitirá a comunicação entre os elementos de rede (de qualquer fabricante que o implemente) e o controlador. O protocolo OpenFlow tem

uma estrutura de mensagens pré-definida, chamadas mensagens OF que são trocadas entre o controlador e o switch (RODRÍGUEZ,2014).

Para cada fluxo que chega, os switches OpenFlow podem executar pelo menos três ações básicas:

- encaminhar o pacote conforme regra da tabela de fluxos: este é o caso geral, pois o controlador é responsável por configurar o fluxo no dispositivo de rede;

- encapsular e enviar o pacote ao controlador por meio de um canal seguro: isto será tipicamente realizado para todos os pacotes que ainda não têm uma regra específica preenchida na tabela de fluxos;

- descartar pacotes: esta função seria usada para impedir o roteamento de determinados fluxos na rede (RODRIGUEZ , 2014).

Para uma melhor compreensão, consideremos a situação descrita (Figura 3): quando o primeiro pacote de um novo fluxo chega a um dispositivo de rede, este ainda não tem pré-configurado um fluxo em seu plano de dados que determine como encaminhar o pacote. Então, o switch envia uma consulta ao controlador, que calcula o melhor caminho para o fluxo específico. Com esta informação e utilizando um protocolo de comunicação apropriado, o controlador configura o fluxo específico no plano de dados de todos os dispositivos intermediários no trajeto, tornando possível o roteamento do pacote. Os sucessivos pacotes do mesmo fluxo, quando já têm pré-configurado o fluxo específico no plano de dados, não terão que realizar uma nova consulta ao controlador.

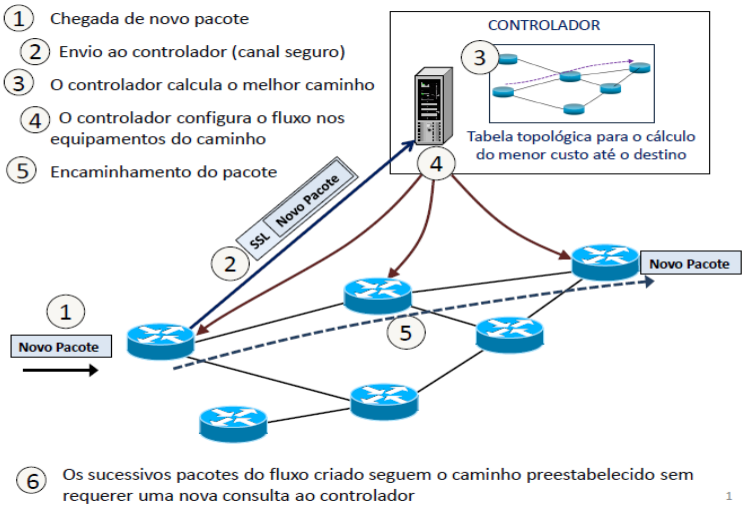

Figura 3 – Funcionamento SDN

Fonte: Rodriguez, 2014, p. 21.

Dependendo das políticas de tráfego criadas pelo controlador, um switch OpenFlow pode se comportar como um roteador, switch normal, firewall ou algum tradutor de endereço de rede.

Embora tenha maior visibilidade por ser o mais utilizado e pesquisado, o OpenFlow não é a única interface *southbound* disponível para SDN. Outro exemplo de protocolo de padronização da troca de mensagens entre os planos de dados e de controle é o *framework* ForCES, proposto pelo IETF.

Os switches comerciais podem habilitar o OpenFlow como recurso. Vários fabricantes já oferecem produtos com a interface OpenFlow: Juniper, NEC, HP, Dell, OpenvSwitch, Cisco, Ciena, entre outros (LARA et al, 2014). Isso pode ser feito através da atualização do firmware desses equipamentos. Através dessa ativação, serão adicionados no switch a tabela de fluxo, o canal seguro e o protocolo OpenFlow.

2.3 - Switch OpenFlow

O switch OpenFlow é o componente básico na camada de infraestrutura. Ele é composto por uma ou mais tabelas de fluxos, uma tabela de grupo, uma tabela de medições e um canal seguro OpenFlow, como mostra a Figura 4.

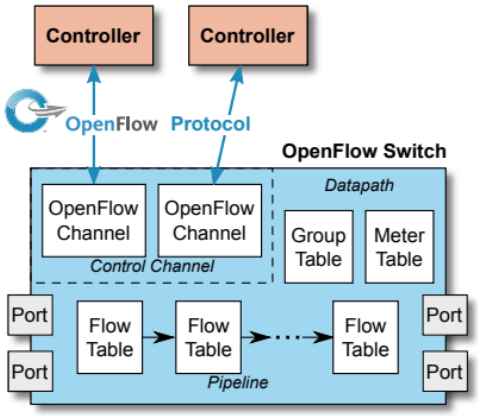

Figura 4: arquitetura de switch OpenFlow

Fonte:Foundation,2015

As tabelas de fluxos são utilizadas para encaminhar o tráfego. A tabela de grupo é um tipo especial de tabela, da qual falaremos mais adiante. A partir da versão 1.3 o OpenFlow passou a suportar o controle das taxas de pacotes através de medidores de fluxos, introduzindo a tabela de medições *Meter Table*. A tabela de medições por fluxo permite ao OpenFlow implementar simples operações de QoS, como limitação de taxas de transmissão e pode ser combinada com as filas por porta para criar políticas de QoS mais complexas.

O canal seguro OpenFlow é a interface por onde são transmitidas as mensagens entre o switch e o controlador. Através dessa interface o controlador configura e gerencia o switch. O mecanismo de segurança padrão do protocolo OpenFlow é o TLS (*Transport Layer Security*). O switch e o controlador estabelecem comunicação através de uma conexão TLS, autenticada e criptografada (FOUNDATION, 2015).

Cada entrada de fluxo na tabela consiste de campos de cabeçalho para identificar o fluxo, contadores, e um conjunto de instruções para aplicar aos pacotes que coincidam com esta entrada, como mostra a Figura 5:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Figura 5 : campos de uma entrada na tabela de fluxo

Fonte: Foundation, 2013

- *Match fields* - dados que serão comparados e combinados com os pacotes. Contém os dados da porta de entrada e cabeçalho dos pacotes;
- *Priority* - prioridade da entrada de fluxo;
- *Counters* - contadores atualizados quando os pacotes são correspondentes à entrada;
- *Instructions* - instruções para modificar o conjunto de ações ou a sequência de processamento;
- *Timeouts* - tempo máximo ocioso antes do fluxo ser considerado expirado pelo switch;
- *Cookie* - contém um valor escolhido pelo controlador, que pode utilizá-lo para filtrar estatísticas, modificações ou deleção de um fluxo. Não é usado no processamento dos pacotes.

Uma entrada de fluxo é identificada pelo *match field* e pela prioridade, que juntos identificam uma entrada de fluxo na tabela.

As ações associadas aos fluxos poderão ser: encaminhar o pacote para uma determinada porta de saída (ou portas), modificar os campos do cabeçalho, ou, como medida de segurança, descartar os dados, entre outras. Além das ações, a arquitetura OpenFlow prevê manutenção de três contadores por fluxo: pacotes, bytes trafegados e duração do fluxo. Esses contadores são implementados para cada entrada da tabela de fluxos e podem ser acessados pelo controlador através do protocolo (FOUNDATION,2015).

Quando um pacote chega, o switch inicia uma busca na primeira tabela, em ordem de prioridade, e pode continuar pelas tabelas seguintes. Se combinar, a ação associada àquele fluxo será executada. Se não, o pacote será descartado ou enviado para processamento no controlador, conforme a configuração de entrada de fluxo. O controlador pode adicionar, atualizar ou excluir entradas na tabela de fluxo, tanto de forma reativa (em resposta a pacotes enviados) quanto de forma proativa.

O protocolo OpenFlow suporta três tipos diferentes de mensagens:

1 – Mensagens Controlador-Switch: geradas pelo controlador para gerenciar e inspecionar o estado de um switch. Exemplos:

- Características (*features*), mensagem enviada quando o controlador requisita as características do switch. O switch deve responder com as características suportadas;
- Configuração (*Configuration*) – mensagem usada para configurar ou solicitar configurações do switch;
- Modificação de estado (*Modify-State*) – mensagem usada para adicionar, deletar, modificar a tabela de fluxos e para configurar propriedades nas portas do switch;
- Envio de pacote (*Send-Packet*) - Utilizado para enviar pacotes por uma determinada porta do switch; dentre outras.

2 - Mensagens Assíncronas: são mensagens que o switch envia para o controlador sem que este tenha solicitado. Essas mensagens são usadas para atualizar o controlador sobre eventos da rede e mudanças no estado do switch. Exemplos:

- Entrada de pacotes (*Packet-In*) - Utilizado quando fluxos não classificados entram no switch;
- Remoção de fluxo (*Flow-Removed*) - Mensagem enviada para o controlador, quando um fluxo é removido da tabela. Seja por *Idle Timeout, Hard Timeout* ou por uma mensagem de modificação da tabela de fluxos que delete a entrada em questão;
- Estado da porta (*Port-Status*) - Mensagem enviada para o controlador sempre que há mudanças nas configurações das portas;

3 - Mensagens Simétricas: podem ser geradas tanto pelo controlador quanto pelo switch. São enviadas sem solicitação. Exemplos:

- Hello - Mensagens trocadas entre o controlador e o switch quando uma conexão é estabelecida;
- Echo - Mensagens usadas para identificação de latência, largura de banda e existência de conectividade.

2.3.1- Tabelas de grupo

OpenFlow usa entradas de fluxo como uma forma de combinar pacotes com os fluxos correspondentes e especificar uma ação para os pacotes que chegam nas interfaces lógicas OpenFlow. A ação especificada em uma ou mais entradas de fluxo pode direcionar os pacotes para uma tabela de grupo. O objetivo do grupo é processar mais rapidamente os pacotes e atribuir uma ação de encaminhamento mais específica a eles.

Um grupo é uma abstração implementada a partir da versão 1.1 do OpenFlow, que o habilita a representar um conjunto de portas como uma única entidade para o encaminhamento de pacotes, facilitando a realização de operações mais complexas com os pacotes, que não poderiam ser facilmente realizadas através de uma entrada na tabela de fluxo. São disponibilizados diferentes tipos de grupos, para representar diferentes abstrações como *multicast* ou *multipathing*.

Cada grupo recebe pacotes como entrada e realiza operações OpenFlow com esses pacotes. Um grupo não é capaz de realizar instruções OpenFlow, por exemplo, não pode enviar um pacote para outras tabelas de fluxos. Ao chegar para a tabela de grupos, os pacotes já foram devidamente tratados pelas tabelas de fluxo, ou seja, já foram combinados com as entradas existentes, uma vez que os grupos não realizam "*matching*" em pacotes - os grupos são apenas mecanismos para realizar ações (ou conjuntos de ações) mais específicas.

O grupo contém listas de ações separadas, e cada lista é chamada de *bucket*. Assim, um grupo contém uma *bucket list*, ou seja, uma lista de listas de ações. Cada *bucket* ou lista de *buckets* contém um conjunto de ações a serem aplicadas na porta de saída do switch; o comportamento depende do tipo de grupo. Alguns tipos de grupos usam parâmetros adicionais junto com o *bucket*. A Figura 6 mostra os componentes de um grupo e de um *bucket*.

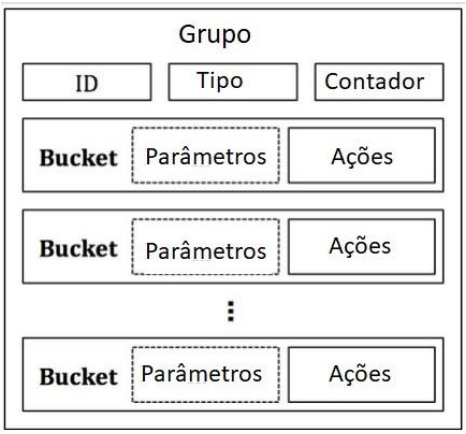

Figura 6: componentes de um grupo e de um bucket

Fonte: Izard,2018

Cada entrada do grupo (Figura 7) é identificada por um identificador de grupo e contém:

Group Identifier	Group Type	Counters	Action Buckets

Figura 7: campos de uma entrada na tabela de grupo

Fonte: Foundation, 2013

- Identificador de grupo: um número inteiro que identifica unicamente o grupo;
- Tipo do grupo: determina a tipo do grupo;
- Contadores: são atualizados quando pacotes são processados pelo grupo;
- *Buckets*: uma lista ordenada de *buckets*, onde cada um contém um conjunto de ações para executar e parâmetros associados.

Existem quatro tipos de grupos: all, select, indirect e fast-failover.

O grupo All é o mais simples, que pega qualquer pacote recebido como input e duplica, para ser operado independentemente em cada *bucket*. Dessa forma, um grupo do tipo All pode ser usado para replicar e operar em diferentes cópias de um pacote definido pelas ações de cada *bucket*. Ações distintas podem ser atribuídas a cada *bucket*, o que permite que operações diferentes sejam aplicadas nas diferentes cópias do pacote. A Figura 8 mostra um grupo do tipo All.

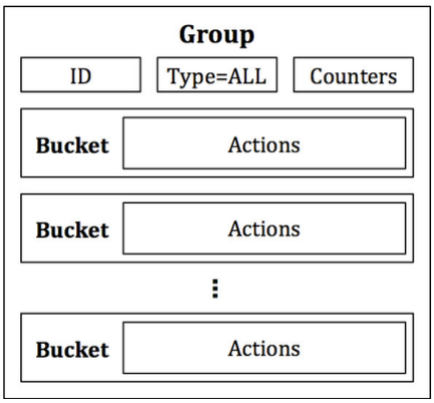

Figura 8 : grupo do tipo All

Fonte: Izard,2018

O grupo Select (Figura 9) foi criado para balanceamento de carga. Cada *bucket* no grupo Select tem um peso associado, e a cada pacote que entra no grupo é enviado para um único *bucket*. O custo de um *bucket* é dado como um parâmetro especial de cada *bucket*. Cada *bucket* em um grupo Select é uma lista de ações, então quaisquer ações suportadas pelo OpenFlow podem ser usadas em cada *bucket*, e assim como no grupo All, os *bucket*s não precisam ser iguais.

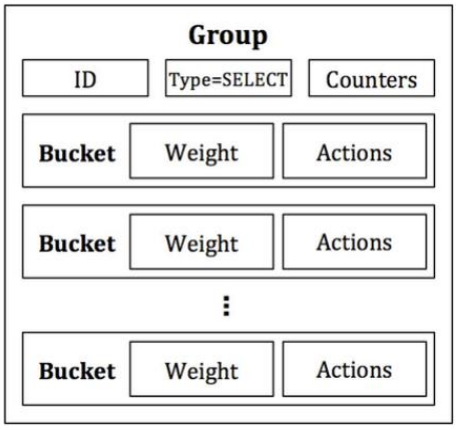

Figura 9: grupo do tipo Select

Fonte: Izard,2018

O grupo Indirect (Figura 10) pode ser difícil de compreender como um grupo, uma vez que contém apenas um único *bucket*, para o qual todos os pacotes são enviados. Em outras palavras o grupo Indirect não contém uma lista de *bucket*s. A função do grupo Indirect é encapsular um conjunto de ações em comum. Por exemplo, se os fluxos A, B e C correspondem a cabeçalhos de pacotes diferentes mas tem em comum um conjunto de ações, esses fluxos podem enviar pacotes para o grupo Indirect, em vez de ter que duplicar a lista de

ações em comum para cada fluxo. O grupo Indirect é usado para simplificar o desenvolvimento e reduzir o consumo de memória de um conjunto de fluxos similares.

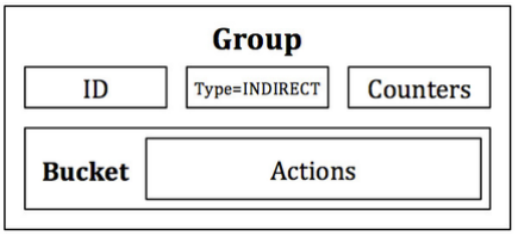

Figura 10: grupo do tipo Indirect

Fonte: Izard,2018

O grupo Fast Failover (Figura 11) é projetado especialmente para detectar e contornar falhas de portas. Assim como os grupos Select e All, o Fast-Failover possui uma lista de *bucket*s. Cada *bucket* possui como parâmetro especial uma porta ou grupo monitorado. A porta/grupo monitorado vai monitorar o status da porta/grupo indicada. Se o status da porta for considerado inativo, então o *bucket* não será usado. Se estiver ativo, então o *bucket* poderá ser usado. Apenas um *bucket* pode ser usado por vez, e o *bucket* em uso não mudará a não ser que status da porta mude para inativa. Quando tal evento ocorre, o grupo fast-failover rapidamente irá selecionar o próximo *bucket* da lista com uma porta que estiver ativa. Neste caso, não há garantia quanto ao tempo de transição para selecionar um novo *bucket*. O tempo de transição depende do tempo de busca para encontrar uma porta/grupo monitorada que esteja ativa, o que depende da implementação do switch. No entanto, a razão para usar um grupo deste tipo é o fato de ser quase garantido que seja mais rápido do que consultar o plano de controle para lidar com o evento de porta inativa e inserir um novo fluxo. Com os grupos fast-failover, a detecção de falhas de links e recuperação ocorre inteiramente no plano de dados.

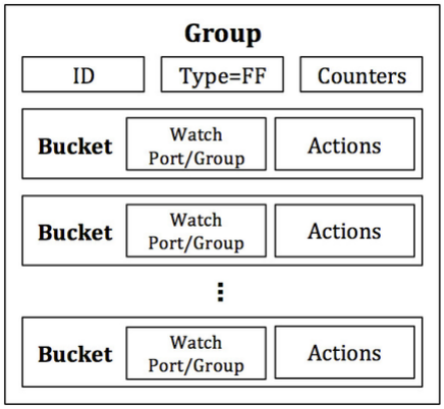

Figura 11: grupo do tipo Fast-Failover

Fonte: Izard,2018

Não é obrigatório que um switch seja capaz de suportar todos os tipos de grupos, apenas os grupos do tipo all e indirect. O controlador também pode consultar o switch sobre quais os grupos suportados.

2.4 - Controlador - *Network Operational System*

O controlador ou *network operational system* é a unidade de controle logicamente centralizada do SDN. Sendo o principal elemento da camada de controle, é a entidade responsável por toda a inteligência da rede. Tem como papéis principais: oferecer abstrações, serviços essenciais e API's para desenvolvedores. Dentre as funções essenciais que um controlador deve desempenhar estão: gerenciamento de dispositivos, estatísticas da rede, mapear a topologia, encaminhamento pelo melhor caminho e mecanismos de segurança.

O controlador SDN pode conter uma API que os serviços podem usar para configurar a rede. Dessa forma, o controlador pode atuar apenas como uma interface para a estrutura de *switching*, enquanto a lógica de controle reside nos serviços que estão usando o controlador. Dependendo do controlador usado, as API's podem ser diferentes. Os controladores e suas API's podem ser adaptados para atender às necessidades no domínio do aplicativo. Um controlador que é projetado e otimizado para data centers, por exemplo, pode não ser adequado para redes de controle no setor elétrico e vice-versa (BOBBA, 2014).

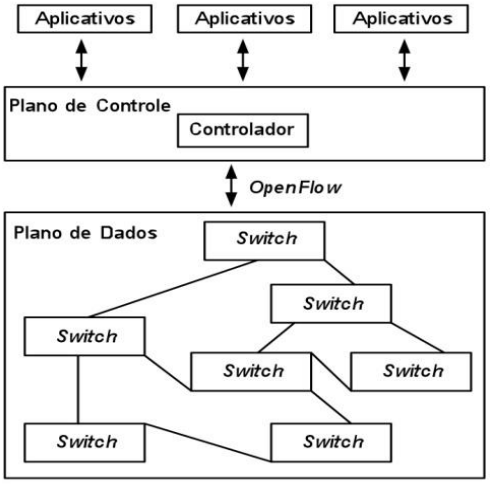

Figura 12: controlador na arquitetura da SDN

Fonte: Bobba, 2014

O controlador é um elemento fundamental na arquitetura SDN por ser a peça principal para o controle lógico realizar configurações de rede baseadas nas políticas definidas pelo administrador da rede. De forma semelhante aos sistemas operacionais tradicionais, a plataforma de controle abstrai os detalhes de baixo nível de conexões e interações com dispositivos de encaminhamento. A maioria dos controladores suporta apenas OpenFlow como *southbound* API.

> É importante abordar o impacto na rede da ocorrência de uma falha do controlador ou da incapacidade de o controlador se comunicar com o dispositivo de rede. Neste caso, o dispositivo de rede continua a operar de forma normal e confiável, enviando todas as comunicações aprovadas. O único impacto para o sistema ocorre quando tem início novas comunicações não configuradas, as quais não serão encaminhadas. (BOBBA et al.,2014,p.6)

O controlador é um software que se conecta de maneira segura ao switch OpenFlow. Através da interface de programação do protocolo o controlador manipula a tabela de fluxos do switch. Existem diversos tipos de controladores com diferentes arquiteturas, que podem ser classificados com base em diferentes aspectos. Classificando do ponto vista da arquitetura, os controladores podem ser centralizados ou distribuídos.

Um controlador centralizado é uma entidade única que gerencia todos os dispositivos de encaminhamento, e tem como limitação a baixa escalabilidade e possibilidade de falhas. Apesar de ser uma entidade logicamente centralizada, o controlador da rede também pode ser um sistema distribuído. Diferente do design centralizado, um controlador distribuído tem potencial para ser escalável e suprir as necessidades de qualquer ambiente.

Um controlador distribuído pode ser um conjunto de nós concentrados em uma localidade ou vários elementos de processamento distantes um do outro. A vantagem de um controlador distribuído é a maior resiliência para falhas físicas e lógicas. No entanto, um desafio é manter o estado da rede sempre atualizado para todos os componentes do controlador. Para a comunicação entre os nós do controlador distribuído, são usadas APIs especiais, chamadas de *Westbound* API e *Eastbound* API. Cada controlador implementa suas próprias API’s desse tipo. Algumas de suas funções são transmissão de dados entre os nós do controlador, algoritmos para a manutenção da consistência dos dados e recursos de monitoramento e notificação – por exemplo, verificar se um dos nós está ligado e notificar

quando um nó assumir o lugar de outro no controle de um conjunto de dispositivos de rede (KREUTZ et al,2015). Um dos fatores mais importantes do protocolo OpenFlow foi permitir que uma arquitetura distribuída pudesse ser aplicada ao controlador da rede. Sem essa possibilidade, o controlador se torna um gargalo, pois à medida que o número de pacotes aumenta, o volume de trabalho do controlador cresce. Através de uma arquitetura distribuída, o controlador pode dividir a carga de trabalho com outros controladores. É importante ressaltar que mesmo distribuído, a entidade controlador é tratada como logicamente centralizada (PANTUZA, 2016).

Com os estudos na área de SDN e OpenFlow se intensificando cada vez mais, novas pesquisas com controladores vão surgindo. No momento, existem mais de 30 controladores OpenFlow diferentes criados para diversas funcionalidades, escritos em diferentes linguagens por empresas, universidades e grupos de pesquisa.

O controlador centraliza a comunicação com os elementos programáveis, oferecendo uma visão unificada da rede. A existência da visão global simplifica a representação dos problemas e a tomada de decisão, facilitando melhor a gestão, a segurança e outros recursos. Com essa visão única pode-se também trabalhar de forma distribuída, através da divisão dos elementos de visão ou através de algoritmos distribuídos (GUEDES et al, 2012). O controlador não necessita estar localizado em um único ambiente físico, já que essa visão unificada da rede é lógica.

2.4.1 – Descoberta de topologia de rede em SDN

Em redes definidas por software, normalmente os controladores usam *Link Layer Discovery Protocol* (LLDP) para descobrir a topologia da rede. LLDP é um padrão aberto usado por dispositivos de rede para descobrir os vizinhos em redes legadas. Em redes SDN, o controlador estrutura o frame LLDP e envia para todas as interfaces de todos os switches. Quando um switch recebe o frame LLDP de seus pares, ele encaminha esse frame para o controlador. Através desse método, o controlador consegue mapear a topologia da rede inteira.

Após construir uma visão de toda a topologia da rede, o controlador, cria uma *spanning tree* usada para prevenir loops durante *flooding* e broadcast. Diferente das redes legadas, o tráfego unicast não segue a *spanning tree*, ele seguirá o menor caminho entre a origem e o destino (HUANG, 2015).

3 – Balanceamento de carga

Balanceamento de carga nas redes é uma tarefa de fundamental importância. Quando muitos pacotes estão trafegando em uma mesma parte da sub-rede, podem ocorrer congestionamentos na rede. Como consequência, ocorrem atrasos e perda de pacotes, levando a uma queda acentuada de desempenho. Um mecanismo de balanceamento de carga deve reduzir o atraso de transmissão e consumo de largura de banda, otimizando a taxa de transferência e portanto a performance da rede.

O balanceamento de carga é uma técnica de engenharia de tráfego que permite distribuir tráfego da rede por mais de um enlace ou por mais de um servidor, de acordo com uma política pré-determinada. Assim, podemos melhorar o desempenho, através do uso otimizado dos recursos disponíveis, buscando a minimizar a latência e maximizar a vazão. O tipo de balanceamento de carga depende do tipo do encaminhamento de pacotes que está configurado nas interfaces do roteador.

3.1- Balanceamento de carga em redes tradicionais

Através de protocolos de roteamento, tais como RIP, IGRP, EIGRP, e OSPF, o roteador aprende várias rotas possíveis em uma rede. Então, o roteador instala na tabela de roteamento a rota com o menor custo.

Quando há mais de um caminho possível, roteadores geralmente usam três tipos de algoritmos para distribuir pacotes pelos links:

1 – balanceamento de carga por destino: significa que o roteador distribui os pacotes com base no endereço de destino. Por exemplo, dados dois caminhos para a mesma rede, todos os pacotes para o host A naquela rede serão enviados pelo primeiro caminho, todos os pacotes para o host B serão enviados pelo segundo caminho, e assim por diante. Esse tipo de balanceamento preserva a ordem dos pacotes, mas pode levar a um uso desigual dos enlaces. Se um host receber a maior parte do tráfego, todos os pacotes enviados a ele usarão o mesmo enlace, deixando outros links com largura de banda sem uso;

2 – balanceamento de carga por fluxo: classifica pacotes em diferentes fluxos, com base em regras de classificação pré-determinadas, tais como 5-tupla IP (endereço de IP da origem, endereço de IP do destino, número do protocolo, número da porta da origem, e número da porta do destino). Assim, pacotes do mesmo fluxo seguem pelo mesmo enlace. No

balanceamento por fluxo, os pacotes podem ser entregues fora de ordem, o que pode ser explicado por fatores como baixa qualidade dos enlaces, ou por pacotes de tamanhos diferentes. Quando pacotes de tamanhos diferentes são transmitidos pelo mesmo link, em situações em que a taxa de transmissão é estável, pacotes de tamanho pequeno podem chegar primeiro mesmo que tenham sido enviados depois de pacotes maiores;

3 – balanceamento de carga por pacotes: os roteadores enviam pacotes em sequência alternada entre os enlaces disponíveis. Por exemplo, o roteador envia um primeiro pacote destinado ao host “A” pelo primeiro enlace, o segundo pacote para o mesmo destino será enviado pelo segundo enlace, e assim por diante. Esse tipo de balanceamento garante distribuição de carga igual por todos os links. No entanto, é possível que esses pacotes cheguem fora de ordem. Balanceamento de carga por pacote distribui o tráfego de forma mais equilibrada do que o balanceamento por fluxo.

Com o paradigma SDN é possível criar um balanceador de carga que apresente maior flexibilidade na criação das regras de balanceamento, tornando o balanceador uma ferramenta programável. Assim, o gerenciamento da rede é simplificado e os administradores de rede dispõem de mais autonomia. Com SDN é possível gerenciar toda a rede através do controlador centralizado, com o qual poderá se distribuir o tráfego de forma mais eficiente, com um custo financeiro menor quando comparado a um balanceador comercial e um melhor desempenho.

3.2 - Balanceamento de carga em SDN

Balanceamento de carga foi uma das primeiras perspectivas de uso para redes definidas por software, e vários algoritmos e técnicas têm sido propostos com esse objetivo. A interface *southbound* OpenFlow pode ser usada para monitorar ativamente a carga no plano de dados. Essa informação pode ser aproveitada para melhorar a utilização dos recursos da rede. Usando algoritmos de otimização e a configuração adequada, é possível atingir os objetivos de latência, performance e tolerância à falhas, junto com a redução do consumo de energia, com o uso de técnicas simples, como desabilitar links e dispositivos em resposta às alterações de tráfego (KREUTZ et al, 2015).

Implementar o paradigma de redes definidas por software permite bastante flexibilidade para inovação e pesquisa e várias abordagens podem ser testadas. Algumas das

estratégias propostas de balanceamento de carga em SDN serão brevemente apresentadas a seguir.

Uma técnica para permitir a escalabilidade dessas aplicações é usar regras-coringa para realizar o balanceamento de carga de forma proativa. As regras-coringa podem ser utilizadas para agregar requisições de clientes, com base nos intervalos de prefixos de IP, por exemplo, permitindo a distribuição e direcionamento de grandes grupos de requisições sem precisar de intervenção do controlador para cada novo fluxo que chegar. Em conjunto, operações em modo reativo ainda podem ser usadas quando grandes volumes de tráfego forem detectados. O controlador precisa monitorar o tráfego da rede e usar algum tipo de limite nos contadores de fluxo para redistribuir os clientes entre os servidores quando gargalos estiverem propensos a acontecer (WANG,2011).

Um método de balanceamento de carga usando vários controladores foi proposto por Yao et al (2015). Nesse algoritmo, o balanceamento de carga é realizado por técnicas de distribuição e centralização. A rede é dividida em clusters compostos por vários switches, tendo os controladores de clusters e um controlador global. Eles projetaram uma abordagem que determina ou detecta sobrecarga nos controladores de cluster e executa balanceamento da carga dentro do cluster ou balanceamento de carga global. Cada controlador de cluster é responsável por decidir como encaminhar os pacotes e o controlador global é responsável pelo balanceamento de carga entre os clusters. Se a carga em um cluster exceder o limite, o balanceamento acontece, transferindo carga de um controlador de cluster sobrecarregado para outro controlador do mesmo cluster, ou se necessário, é enviada uma requisição ao controlador global para realizar um balanceamento de carga, distribuindo fluxos entre os clusters.

Gholami e Akbari (2016) apresentam um método para controlar o congestionamento em data centers e, dessa maneira, otimizar a performance da rede. No método proposto, o reconhecimento de congestionamentos nos links é descoberto através da constante verificação de estatísticas das portas dos switches. Ao verificar um fluxo com mais de 70% de uso, com base nas estatísticas correntes da rede, é iniciado o recálculo de um novo caminho com mais recursos livres.

De forma similar, Hertiana et al (2015) propõem um balanceamento de carga utilizando Openflow com uma combinação de diferentes rotas aliada com um recurso para adaptação da taxa de entrega. Pode-se concluir que essa combinação produz uma melhor

performance que o modelo tradicional de único caminho e o modelo de diferentes rotas sem adaptação da taxa de entrega.

Os autores Song et al (2016) propõem um algoritmo para controlar fluxos de congestionamentos em redes SDN. Busca-se prever e inferir na rede antes da ocorrência de perda de pacotes. Para esse propósito, ao atingir 70% do uso da rede a proposta recalcula uma nova rota para o tráfego dos dados com base na utilização das portas do switch. Observa-se que a solução é capaz de reduzir os congestionamentos em ambientes de alto congestionamento. Com isso, seria possível substituir o *slow start* do algoritmo de controle de congestionamento TCP pelo algoritmo proposto.

O Omniscient TCP (OTCP), apresentado por Jouet et al (2016), é uma abordagem que utiliza a SDN para computar parâmetros de controle de congestionamento específicos do ambiente baseado nas propriedades disponíveis da rede que se encontram no controlador. Voltado para rede interna de data centers, o objetivo é abordar as deficiências de TCP sob cargas de trabalho agregadas, alcançando alta taxa de transferência e latência baixa e estável. Para isso, realiza configurações dos parâmetros de congestionamento do TCP para cada rota ponto a ponto considerando a topologia, latência, taxa de transferência e o buffer.

Estas são algumas das recentes pesquisas abordando o paradigma de redes definidas por software, focadas em um dos aspectos mais importantes da performance da rede, o balanceamento de carga.

3.3 – Algoritmos de balanceamento de carga

O roteamento por múltiplos caminhos tem como objetivo explorar os recursos da rede subjacente ao prover múltiplos caminhos entre a origem e o destino. O roteamento multicaminhos tem o potencial de aumentar a disponibilidade de largura de banda nos enlaces, permitindo à rede suportar um tráfego de dados superior ao que poderia utilizando um único caminho.

Dois aspectos se destacam em um algoritmo de roteamento multicaminhos: computar todos os caminhos possíveis que não contenham *loop*, e dividir o tráfego entre estes caminhos descobertos.

Como exemplo de soluções largamente usadas para prover roteamento multicaminhos, destacamos o *Equal-Cost-MultiPath* (THALER, 2000, *apud* REZENDE,

2016), que determina a interface de saída de um fluxo por meio do *hash* do cabeçalho dos pacotes. Assim, mesmo que diferentes fluxos entre as mesmas entidades fim-a-fim sigam caminhos distintos, todo o tráfego para um dado fluxo tende a seguir o mesmo caminho. Já o *MultiPath* TCP (FORD et al. 2013, *apud* REZENDE, 2016), divide um fluxo TCP em sub-fluxos, o que permite que cada sub-fluxo seja, do ponto de vista lógico, transmitido por um caminho distinto. Contudo, o MPTCP é centrado nos sistemas finais e mesmo alterando a pilha de protocolo destes elementos, não há garantia alguma de que estes sub-fluxos serão, de fato, encaminhados através de multicaminhos disjuntos do ponto de vista físico (REZENDE et al,2016).

Como o objetivo desta pesquisa é mostrar a aplicação de conceitos OpenFlow no balanceamento de carga em SDN, vamos apresentar uma estratégia de balanceamento de carga usando multicaminhos. A estratégia selecionada é o *Multipath Routing Load Balancing*, proposta por Wildan (2018).

3.3.1 - Algoritmo *Multipath Routing Load Balancing*

Ao inicializar a aplicação, é realizada a descoberta de caminho. Para isso o algoritmo utiliza a busca em profundidade, que segue uma política de visitar sempre os nós mais profundos primeiro, para somente depois retroceder (*backtracking*). Ao retroceder, ele encontra os outros possíveis nós e armazena em pilha. Assim, a busca em profundidade permite encontrar todos os possíveis caminhos entre dois nós em uma rede.

O algoritmo de busca em profundidade retorna uma lista de caminhos possíveis, então é necessário atribuir um custo a cada caminho. O custo dos links é calculado usando a fórmula "função de custo". É utilizada a mesma função de custo empregada no protocolo *Open Shortest Path First* (OSPF):

$$C = \frac{Bref}{B_L}$$

onde B_{ref} é a largura de banda de referência e B_L é a largura de banda do enlace.

Isso significa que quanto maior a largura de banda do enlace, menor será o custo desse link. Após calcular o custo de cada link, totaliza o custo do caminho todo. Mais adiante, outro cálculo de custo será realizado para medir os custos dos *bucket*s.

O próximo passo é instalar os caminhos encontrados nos switches. A função de instalar caminhos realiza as etapas:

1. Lista os caminhos possíveis da origem ao destino;

2. Faz um loop por todos os switches que pertencem ao caminho:

a. Lista todas as portas do switch que pertencem ao caminho

b. Se mais de uma porta do switch contém um caminho, é criada uma tabela de grupo do tipo "Select". Se não, apenas instala uma entrada na tabela de fluxo normalmente. Ao criar uma tabela de grupo, são criados *bucket*s, que são grupos de ações, onde são especificados:

- *bucket weight*: o custo do *bucket*;
- *watch port*: a porta a ser monitorada (não obrigatória para um grupo do tipo "select"),
- *watch group:* outra tabela de grupos a ser monitorada (não obrigatória),
- *actions*: a ação do OpenFlow a ser realizada;

Para calcular o custo do *bucket*, a seguinte fórmula é utilizada:

$$bw(p) = \left(1 - \frac{pw(p)}{\sum_{i<n}^{i=0} pw(i)}\right) x10$$

em que para um caminho ***p*** :

- ***bw*** é o custo do *bucket*, ***0 ≤ bw(p) < 10;***
- ***pw*** é o custo do caminho, (utiliza a função de custo $\frac{Bref}{B_L}$ calculada anteriormente);
- ***n*** é o número total de caminhos disponíveis.

Basicamente, essa fórmula calcula a razão entre o custo do caminho "p" e o custo total de todos os caminhos possíveis. Simplificando, quando se trata de custo do caminho, em que se pretende utilizar o SPF, então o menor é o melhor. Porém no contexto de *bucket*s nas tabelas de grupo OpenFlow, a prioridade é escolher um *bucket* com o maior custo, então

quanto maior o custo, melhor o caminho. Usando esta fórmula, quanto menor o custo do caminho, então maior será seu custo do *bucket*.

4 - Implementação de uma rede SDN e testes

Neste capítulo, são apresentadas as ferramentas usadas para criar um ambiente de testes e será demonstrada a estratégia de balanceamento de carga apresentada.

4.1 – Emulador de redes Mininet

O ambiente utilizado para construção da rede SDN foi o emulador de redes Mininet. Desenvolvido por pesquisadores da Universidade de Stanford, Mininet é uma ferramenta amplamente utilizada para pesquisas, desenvolvimento, prototipação, realização de testes, e quaisquer aplicações que necessitem de uma rede experimental completa.

Mininet é um emulador de redes que cria uma rede virtual composta por hosts, switches, controladores e links. Os hosts da Mininet rodam o sistema Linux padrão e os switches suportam OpenFlow.

O Mininet é um emulador do tipo CBE (*Container-Based Emulation*) que emprega a virtualização a nível de processo, uma forma mais leve de virtualização onde muitos recursos do sistema são compartilhados. Compartilhando recursos como tabela de paginação, estruturas de dados do kernel e sistema de arquivos, esse tipo de emulação alcança maior escalabilidade que outras que realizam uma emulação completa, ou seja, usando uma máquina virtual para cada dispositivo da rede (KLEIS,2015). A Figura 13 mostra a arquitetura da Mininet.

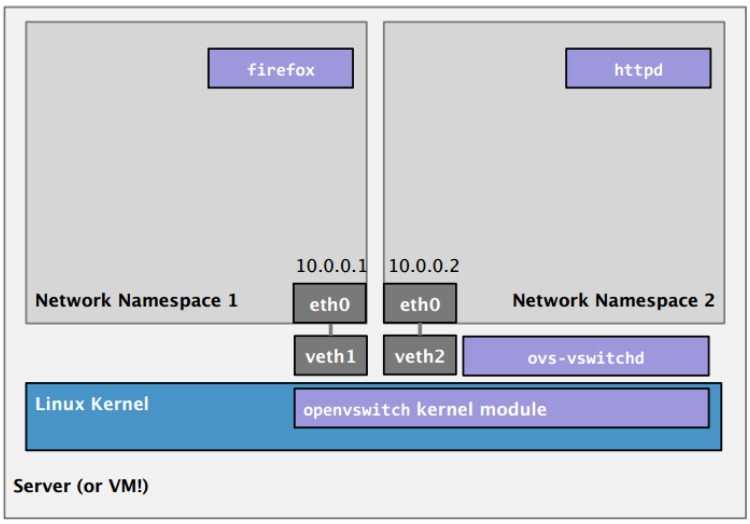

Figura 13: Mininet

Fonte: Kleis, 2015

O Mininet é capaz de emular links, hosts, switches e controladores, utilizando processos que rodem em *network namespaces* e pares Ethernet virtuais (KLEIS,2015):

- Links: um par Ethernet virtual atua como um cabo conectando duas interfaces virtuais. Pacotes enviados através de uma interface são entregues na outra e cada interface se comporta como uma interface Ethernet completa e funcional para todo o sistema e aplicação.
- Host: é um processo do shell movido para o seu próprio espaço de nome da rede, ou seja, cada host apresenta uma instância da interface de rede independente. Cada dispositivo apresenta uma ou mais interfaces virtuais e um *pipe* para um processo pai do Mininet (mn), que envia comandos e monitora a saída.
- Switches: switches OpenFlow são remotamente configurados e gerenciados pelo controlador, podendo ser configurados para agirem como switches convencionais.
- Controladores: o controlador pode ser nativo do Mininet ou um controlador remoto.

Para controlar e gerenciar todos os dispositivos emulados, o Mininet fornece uma CLI (*Command Line Interface*) conhecedora de toda a rede, ou seja, através de um único console, é possível controlar todos os dispositivos emulados. Outra grande vantagem desse emulador é a facilidade de criação de topologias customizadas através de uma API (*Application Programming Interface*) para programação em Python.

4.2 - Controlador Ryu

Como visto anteriormente, o controlador é o cérebro nas redes definidas por software, já que é responsável por definir a lógica de encaminhamento através das regras que irão configurar nos switches OpenFlow. O controlador utilizado nesse trabalho foi o Ryu (Figura 14).

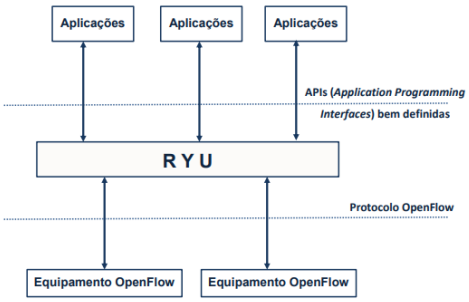

Figura 14 : Controlador Ryu

Fonte: Rodriguez, 2014

Dentre os principais componentes da arquitetura do Ryu (Figura 15) destacam-se as bibliotecas, que dão suporte às várias operações de processamento de pacotes. O Ryu suporta todas as versões do protocolo OpenFlow, e inclui uma biblioteca de codificador e decodificador do OpenFlow. Um dos principais componentes da arquitetura é o OpenFlow-Controller, que é responsável por gerenciar os switches, configurar fluxos, escutar eventos e etc.

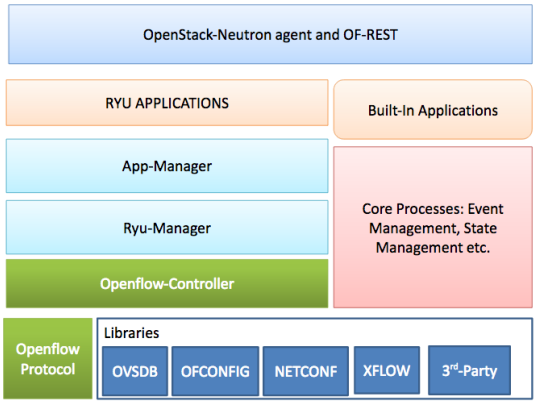

Figura 15: arquitetura do controlador Ryu

Fonte: Rao,2014

O Ryu-manager é o principal executável. Quando executado, escuta um endereço de IP especificado numa determinada porta (6633 por default). Então, qualquer switch OpenFlow pode se conectar ao Ryu-manager. A aplicação manager é o componente fundamental para todas as aplicações que o Ryu pode executar, pois todas as aplicações utilizam a classe RyuApp do manager. O processo principal da arquitetura inclui gerenciamento de eventos, troca de mensagens, gerenciamento de memória, dentre outros.

Ryu *Northbound*: na camada API, o Ryu possui um plug-in Openstack Neutron que suporta configurações VLAN. O Ryu também suporta uma interface REST para as operações OpenFlow.

As aplicações do Ryu são entidades *single-thread* que implementam várias funcionalidades, tais como *firewall*, topologia, VLAN, dentre outras. As aplicações do Ryu enviam eventos assíncronos entre si.

4. 3 - Open vSwitch

Switches virtuais são muito diferentes dos switches tradicionais. Os switches físicos usam uma tabela TCAM (*ternary content-addressable memory*) para conseguir uma alta velocidade de classificação e encaminhamento de pacotes. Já os switches virtuais usam vários níveis de caches e *caching* de fluxos para atingir altas taxas de encaminhamento.

Open vSwitch é um switch virtual que suporta OpenFlow, capaz de lidar com virtualização de redes avançada, realizando processamento de pacotes com reduzidos recursos do hipervisor. O Open vSwitch fornece uma interface que permite a configuração dos switches virtuais, similar às interfaces de gerenciamento dos switches físicos. Também possibilita a manipulação remota do plano de dados de seus switches virtuais (tabelas de fluxos), especificando como processar os pacotes com base nos cabeçalhos de camadas 2, 3 e 4. A interface para a manipulação das tabelas é implementada por meio do protocolo OpenFlow (RODRIGUEZ,2014).

A arquitetura do Open vSwitch é composta de três componentes: vswitchd, ovsdb-server e o datapath:

- vswitchd é o principal processo *daemon* no espaço de usuário, o ovs-vswitchd é responsável por ler a configuração do Open vSwitch do ovsdb-server e repassar esta configuração para os *bridges*. Também transmite alguns dados

referentes a status e estatísticas dos *bridges* para o banco de dados. A maior complexidade do OVS está no ovs-vswitchd, onde todas as decisões de encaminhamento e protocolos de rede são tratados;

- ovsdb-server é o servidor de banco de dados do Open vSwitch. Algumas configurações voláteis, como fluxos, são armazenadas no datapath e no vswitchd. Configurações persistentes são armazenadas no ovsdb, que permanecem mesmo após uma reinicialização. O ovsdb-server também oferece interfaces para o OVSDB se conectar com sockets;
- datapath é o módulo do kernel que realiza o encaminhamento dos pacotes. Datapath é principal módulo de encaminhamento de pacotes do OVS, implementado para obter alta performance. Ele armazena fluxos OpenFlow, e executa as ações nos pacotes correspondentes (CHIAO,2016).

Quando o Open vSwitch é iniciado, dois serviços são instanciados: ovs-vswitchd e ovsdb-server. O ovs-ovswitchd recebe as mensagens OpenFlow do controlador. A comunicação entre o ovs-ovswitchd e o datapath se dá através do netlink, um socket similar ao Unix Domain Socket. A figura 16 mostra a arquitetura do Open vSwitch.

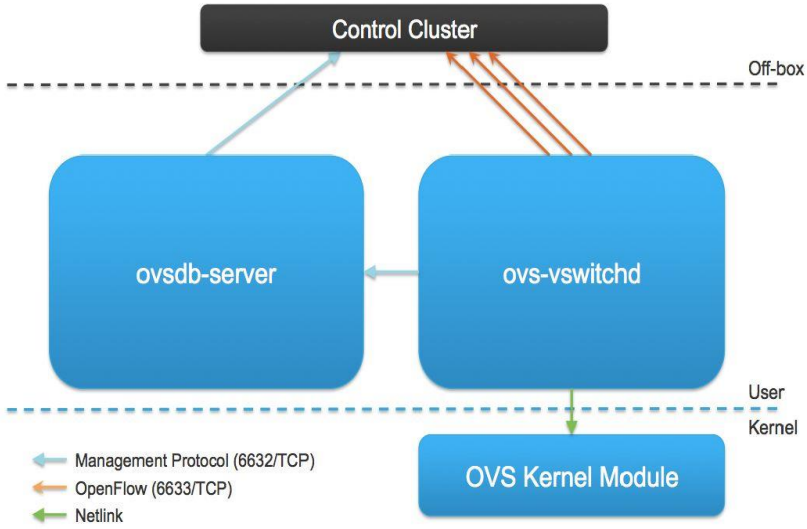

Figura 16 : arquitetura do Open vSwitch

Fonte: Chiao,2016

O Open vSwitch tem três camadas de tabelas de buscas, como mostrado na Figura 17.

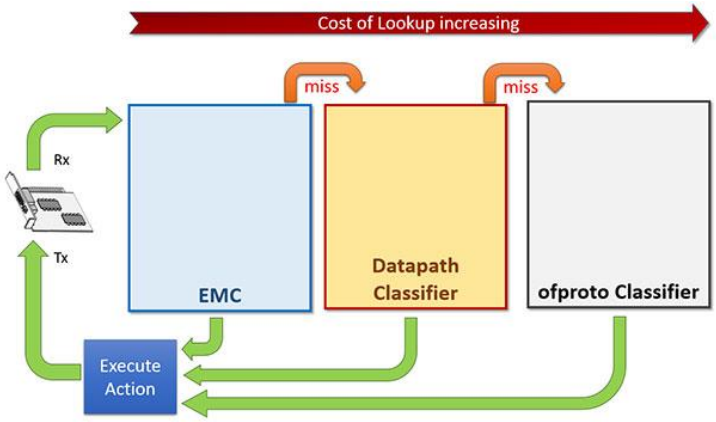

Figura 17 : tabelas de busca do Open vSwitch

Fonte: Fischetti,2016

Quando um pacote é recebido pelo módulo kernel, é primeiramente comparado com as entradas da tabela *Exact Match Cache* (EMC), que é o primeiro e mais rápido mecanismo que o OVS usa para determinar o quê fazer com um pacote que acaba de chegar. Se uma entrada correspondente é encontrada, então as ações associadas (por exemplo, modificar os cabeçalhos ou encaminhar os pacotes) são executadas no pacote. Caso não corresponda a nenhuma entrada dessa tabela, o pacote é enviado para o *datapath classifier* ou dpcls. O dpcls é implementado como um espaço de tuplas que suporta *matching* bit a bit nos campos do cabeçalho do pacote. Caso não encontre correspondência no dpcls, os pacotes são enviados para o processo ovs-vswitchd, também chamado de *ofproto classifier*, configurado pelo controlador, que então executa no pacote a sequência de procedimentos previstos pelo OpenFlow. Após executá-los, o ovs-vswitchd devolve o pacote para o espaço do kernel para ser devidamente encaminhado, e instala uma entrada no cache de fluxos, de modo que os pacotes pertencentes ao mesmo fluxo não precisarão realizar novamente esse procedimento custoso (FISCHETTI et al,2016).

A figura 18 mostra o ciclo de vida de um novo fluxo ao chegar aos componentes de encaminhamento do Open vSwitch: o primeiro pacote não corresponde a nenhuma entrada e é enviado ao espaço de usuário para posterior processamento. Os pacotes subsequentes serão processados inteiramente no kernel.

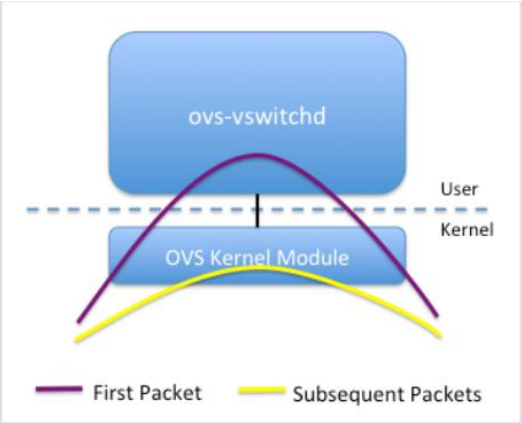

Figura 18: pacote no Open vSwitch

Fonte: Fischetti et al,2016

4.4 – Testes realizados e resultados

Como a proposta dessa monografia é apresentar as redes definidas por software, mostrando a aplicação dos conceitos do OpenFlow no balanceamento de carga, o teste realizado busca implementar uma rede, gerar tráfego, e analisar o desempenho do algoritmo de balanceamento *Multipath Routing*, apresentado anteriormente.

Para a realização dos testes, foi criada na Mininet uma topologia redundante com quatro switches e dois hosts, controlados pelo controlador remoto Ryu. Todos os links foram definidos com as mesmas características, sendo bw - largura de banda de 100Mbps. Outros parâmetros como perda do link (*loss*), *delay* e tamanho máximo da fila permanecem como default da Mininet.

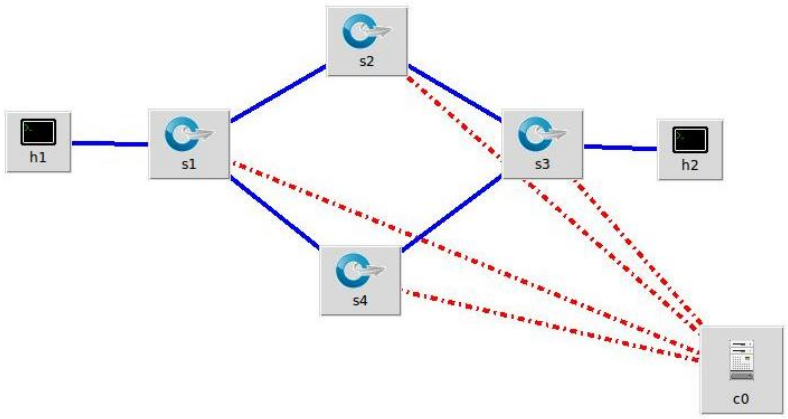

Figura 19 :Topologia utilizada

Como visto na Figura 19, a partir switch s1, que está conectado ao host 1, existe mais de um caminho possível para alcançar o host 2: s1 – s2 – s3 ou s1 – s4 – s3. Da mesma forma, do host 2 para atingir o host 1.

Primeiramente, o emulador Mininet é inicializado com a topologia personalizada criada. Então o controlador remoto é inicializado.

Ao se inicializar a aplicação, a primeira ação que é realizada é a descoberta da rede pelo controlador. O algoritmo utiliza a busca em profundidade, que retorna uma lista de caminhos possíveis, então é necessário atribuir um custo a cada caminho. O custo dos links é calculado usando a função de custo, já apresentada anteriormente.

Após a topologia ser salva, com a descoberta de todos os switches e a atribuição dos pesos, o a próximo passo é instalar os caminhos encontrados nos switches.

Através do comando *dump-flows*, é possível ver as entradas de fluxo que foram instaladas nas tabelas. Ao inicializar, não temos entradas instaladas (Figura 20).

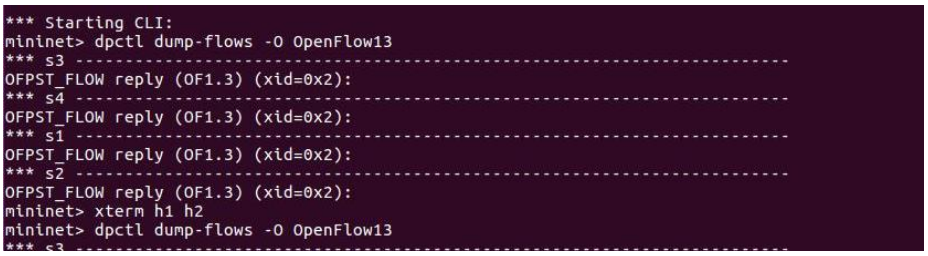

Figura 20 : tabelas de fluxo vazias ao inicializar

O comando ping é utilizado para testar a rede. O ping é iniciado do Host 2 para o Host 1. A tabela de fluxos do switch não possui nenhuma entrada de fluxos, portanto o pacote é enviado para o controlador. O agente Openflow residente no switch consulta o controlador quando pacotes são recebidos mas não há uma entrada de fluxo referente.

Como visto na Figura 17, a partir switch s1, que está conectado ao host 1, existe mais de um caminho possível para alcançar o host 2: s1 – s2 – s3 ou s1 – s4 – s3. Ou seja, há mais de uma porta disponível para saída do tráfego, portanto, é instalado um grupo de ações (Figura 21).

Figura 21: fluxos instalados na tabela do switch 1

O mesmo ocorre no switch 3, qe está conectado ao Host 2, e possui mais de um caminho possível para atingir o Host 1 (Figura 22):

Figura 22: fluxos instalados na tabela do switch 3

As entradas de fluxo que contém o campo “actions = group:” significam que a regra aplicada para esse fluxo é encaminha-lo para a tabela de grupo com o id especificado. Através do comando *dump-groups*, é possível verificar as entradas nas tabelas de grupo descrevendo suas respectivas ações:

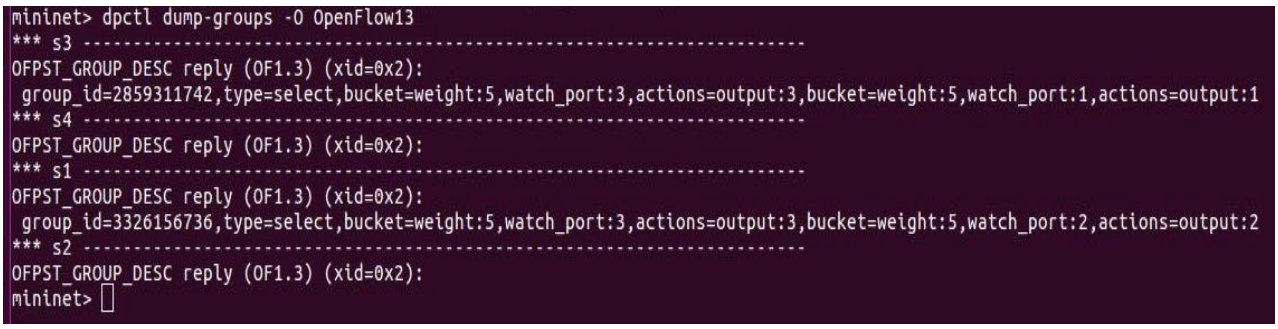

Figura 23: entradas na tabela de grupo

Na Figura 23, vemos que cada entrada na tabela de grupo contém o id do grupo, o tipo do grupo, que neste caso é do tipo select, o peso do *bucket*, a porta sendo monitorada, e a ação, que é encaminhar o pacote pela respectiva porta monitorada.

O peso, ou custo de cada *bucket*, representa o custo de enviar os pacotes através daquela porta. Neste caso, os *bucket*s tem o mesmo peso = 5. O custo é calculado utilizando a fórmula de custo já apresentada anteriormente.

No caso dos switches em que há apenas um caminho possível, não são criadas ações de grupo, é instalada apenas uma entrada comum na tabela de fluxo (Figura 24).

```
*** s4 ----------------------------------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=40.203s, table=0, n_packets=89, n_bytes=5340, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:
65535
 cookie=0x0, duration=6.742s, table=0, n_packets=0, n_bytes=0, ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:1
 cookie=0x0, duration=6.742s, table=0, n_packets=1, n_bytes=42, priority=1,arp,arp_spa=10.0.0.1,arp_tpa=10.0.0.2 actions=output:1
 cookie=0x0, duration=6.742s, table=0, n_packets=0, n_bytes=0, ip,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:2
 cookie=0x0, duration=6.742s, table=0, n_packets=1, n_bytes=42, priority=1,arp,arp_spa=10.0.0.2,arp_tpa=10.0.0.1 actions=output:2
 cookie=0x0, duration=27.537s, table=0, n_packets=1, n_bytes=203, priority=1,ipv6 actions=drop
 cookie=0x0, duration=40.236s, table=0, n_packets=5, n_bytes=389, priority=0 actions=CONTROLLER:65535
*** s1
```

Figura 24: fluxos instalados na tabela do switch 4

Para gerar tráfego para os testes, foi utilizado o Iperf. Com o Iperf, é possível configurar um host para atuar como servidor, escutando em uma ou mais portas e outro host como cliente, que enviará tráfegos para o servidor. Definiu-se que os pacotes enviados seriam TCP, para gerar tráfegos “ratos”, que são fluxos formados por uma pequena quantidade de pacotes, um tipo de fluxo que ocorre com grande frequência.

No host 1 é iniciado uma instância do Iperf-server, e no host 2 são iniciados 5 instâncias do Iperf-client. O objetivo é que cada par cliente-servidor gere um fluxo diferente, pois terão números de porta de origem diferentes.

No switch 3, a porta 2 recebe o tráfego gerado pelo Host 2, e deve enviá-los pelas portas 1 e 3 (Figura 25).

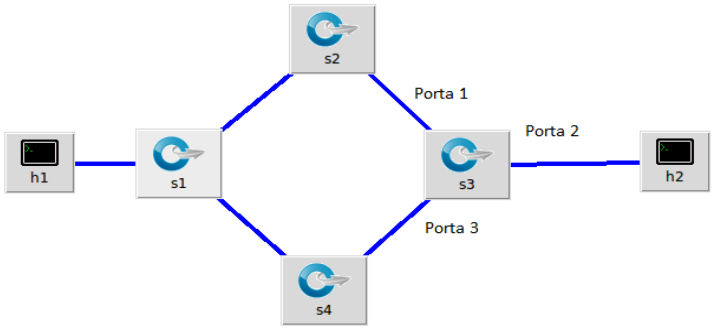

Figura 25 : as portas do switch 3 por onde o tráfego será encaminhado

Como já foi mostrado, cada *bucket* possui seu custo. Para cada pacote encaminhado, o switch seleciona um *bucket* e executa suas ações. Cada *bucket* tem uma regra de encaminhamento para um caminho diferente (porta L1 diferente).

Examinando a taxa de pacotes transmitidos pelas portas 1 e 3, tem-se que:

- Porta 2 – recebidos (rx) = 452.719

- Porta 3 – transmitidos (tx) = 263.678
- Porta 1 – transmitidos (tx) = 189.288

```
*** s3 ----------------------------------------------------------------------
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port  1: rx pkts=175213, bytes=11567059, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=189288, bytes=939718387, drop=0, errs=0, coll=0
           duration=1720.128s
  port LOCAL: rx pkts=0, bytes=0, drop=1, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=2248.734s
  port  2: rx pkts=452719, bytes=2428123950, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=416491, bytes=27491559, drop=0, errs=0, coll=0
           duration=1720.128s
  port  3: rx pkts=241407, bytes=15935201, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=263678, bytes=1488426209, drop=0, errs=0, coll=0
           duration=1720.130s
*** s4 ----------------------------------------------------------------------
```

Figura 26 : saídas balanceadas nas portas do Switch

O mesmo teste foi realizado em uma topologia maior, que oferecia mais caminhos possíveis para o host 2 alcançar o host1, como mostra a Figura 27 .

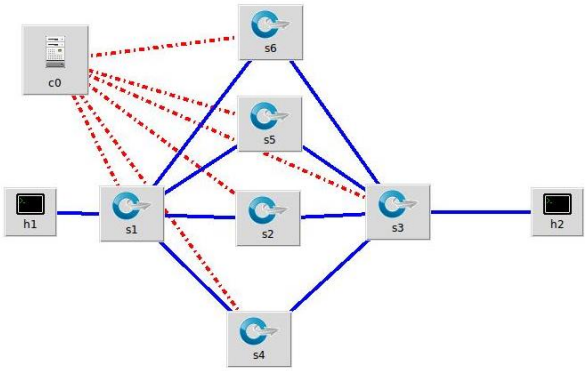

Figura 27 : topologia com 4 caminhos possíveis entre os hosts

Examinando a taxa de pacotes transmitidos pelas portas 1,3,4 e 5, vemos que o balanceamento ocorreu de forma mais irregular (Figura 28):

- Porta 2 – recebidos (rx) = 806.142
- Porta 3 – transmitidos (tx) = 624
- Porta 1 – transmitidos (tx) = 624
- Porta 4 - transmitidos (tx) = 721.432
- Porta 5 - transmitidos (tx) = 85.957

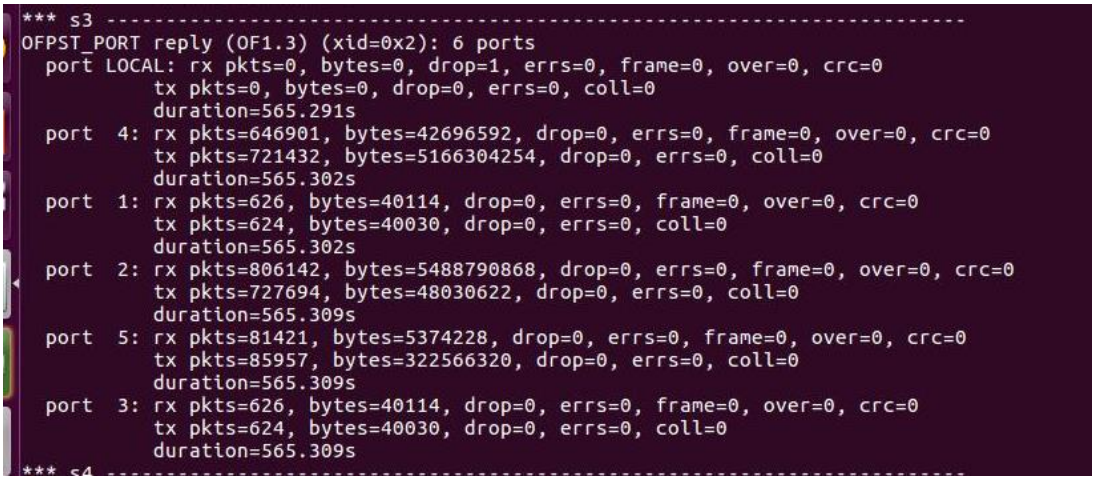

Figura 28 :saídas balanceadas nas portas do Switch3

Vimos que o balanceamento de carga é realizado utilizando o conceito OpenFlow de tabelas de grupo do tipo Select. Um grupo do tipo Select executa um *bucket* do grupo, balanceando os pacotes entre os *bucket*s de acordo com seus respectivos custos. Considerando que em ambos os cenários de testes, os buckets tinham o mesmo peso, se faz necessário entender como o Open Vswitch divide os pacotes entre os buckets.

Como foi mostrado anteriormente, o Open Vswitch tem uma arquitetura com três camadas de tabelas de buscas. Quando um pacote é recebido pelo módulo kernel, é primeiramente comparado com as entradas da tabela *Exact Match Cache* (EMC). Se uma entrada correspondente é encontrada, então as ações associadas são executadas no pacote. Caso não corresponda a nenhuma entrada dessa tabela, o pacote é enviado para o *datapath classifier* ou dpcls. Caso não encontre correspondência no dpcls, os pacotes são enviados para o processo ovs-vswitchd, também chamado de *ofproto classifier*, configurado pelo controlador, que então executa no pacote a sequência de procedimentos previstos pelo OpenFlow. Após executá-los, o ovs-vswitchd devolve o pacote para o espaço do kernel para ser devidamente encaminhado. Para otimizar a performance da varredura nessas tabelas, elas são implementadas como tabelas *hash*.

O Open Vswitch aplica uma função *hash* (MurmurHash, função não criptográfica geralmente usada para buscas baseadas em *hash*) tanto nos campos da regra, quanto nos campos do cabeçalho do pacote, obtendo como *output* dessa função um valor-hash. Este valor será multiplicado pelo peso do bucket, o valor resultante é chamado de *score*. Então, será selecionado o bucket que tiver maior *score*. Por default, o referido valor-hash utilizado

corresponde a um *hash* simétrico computado sobre a combinação de endereços MAC, tags da VLAN, Ether type, endereço IP e números de porta L4.

Assim, o valor resultante da função de hash aplicado aos campos do cabeçalho, termina por influenciar na seleção dos buckets e portanto no balanceamento do tráfego na rede.

Por default, o referido valor-hash utilizado corresponde a um hash simétrico computado sobre a combinação de endereços MAC, tags da VLAN, Ether type, endereço IP e números de porta L4.

Os campos “hasheados” se tornam exatamente correspondentes aos fluxos no datapath do switch. Por exemplo, se a porta de origem TCP é “hasheada”, os fluxos criados irão corresponder ao valor da porta de origem TCP presente no pacote recebido. Como cada conexão TCP tinha diferentes valores de porta de origem, um fluxo diferente será criado para cada conexão TCP, portanto será “hasheado” para um *bucket* de um grupo Select.

Em virtude da natureza da implementação, os *bucket*s são selecionados arbitrariamente, pelo hash de alguns campos do pacote, proporcionalmente aos custos. Neste caso, vimos que os *bucket*s possuíam o mesmo custo. Portanto, temos que o tráfego foi balanceado de forma mais equilibrada entre os *bucket*s, em uma topologia menor.

5 – Conclusão

Este trabalho teve como objetivo apresentar um estudo sobre o paradigma de redes definidas por software, e realizar um estudo de caso mostrando a aplicação de conceitos OpenFlow no balanceamento de carga em SDN. Para isso, uma rede SDN foi implementada com o emulador de redes Mininet e o controlador Ryu, onde foi executado o algoritmo de balanceamento *Multipath Routing* (WILDAN,2018).

Buscando fundamentar o entendimento do paradigma de redes definidas por software, mostrou-se que esta arquitetura possibilita maior facilidade e controle para a manutenção de redes complexas, permitindo superar as limitações da infraestrutura tradicional de redes. As redes definidas por software são baseadas na desagregação o plano de dados do plano de controle. Assim, a rede se estrutura com roteadores e switches que pertencem ao plano de dados, enquanto no plano de controle tem o controlador centralizado e programável, que detém uma perspectiva completa da rede, em tempo real.

Dessa forma, os proprietários do sistema têm uma visão global de toda a rede, podendo monitorá-la como um único ativo. Esta visualização centralizada torna possível o gerenciamento de cenários de operação de todo o sistema, fornecendo dados sobre quais fluxos são permitidos, onde eles estão sendo efetuados, e qual o caminho usado para chegar ao destino.

Com a arquitetura SDN a visualização da rede não fica limitada ao L2 ou L3; ela não é limitada às camadas. Os administradores da rede terão a capacidade de projetar todos os enlaces virtuais em que todos os fluxos de comunicação vão trafegar, pré-configurar as ações de resposta aos eventos, monitorar os fluxos de comunicação e reagir ao desempenho indesejado para manter os sistemas críticos operacionais. As redes definidas por software permitem ao administrador uma compreensão eficiente do que aconteceu, sendo possível definir quais enlaces são atingidos, desde qual fio foi cortado até qual segmento de rede está experimentando um ataque de negação de serviço (DoS: "*Denial-of-Service*").

A separação entre o plano de controle e o plano de dados é realizada por meio de uma API (*Application Programming Interface*). O exemplo mais famoso é o OpenFlow, que foi proposto para uniformizar a comunicação entre os switches e o controlador na arquitetura SDN. Portanto, desempenha papel fundamental nesta arquitetura. Ao atuar como uma

interface *southbound* entre as camadas de controle e dados, ele implementa as regras de encaminhamento que tornam o conceito de SDN possível.

Balanceamento de carga foi uma das primeiras perspectivas de uso para redes definidas por software, e vários algoritmos e técnicas têm sido propostos com esse objetivo. A interface *southbound* OpenFlow pode ser usada para monitorar ativamente a carga no plano de dados. Essa informação pode ser aproveitada para melhorar a utilização dos recursos da rede.

A fim de mostrar os conceitos OpenFlow realizando o balanceamento de carga, foi implementada uma SDN utilizando o emulador Mininet e o controlador remoto Ryu. Como visto, uma topologia simples com quatro switches e dois hosts foi utilizada. Propositalmente, existem dois caminhos de mesmo custo entre os dois hosts. Fluxos TCP são gerados pelo host cliente para o servidor e o tráfego é balanceado uniformemente utilizando os dois caminhos possíveis. O balanceamento acontece da seguinte forma: o pacote é encaminhado para uma tabela de grupo Select, em que cada *bucket* tem um peso. Para cada pacote encaminhado, o Open Vswitch escolhe um *bucket* e executa as suas respectivas ações. Cada *bucket* contém uma regra de encaminhamento para um diferente nó vizinho, ou seja, para uma porta L1 diferente. Os *buckets* são selecionados arbitrariamente, usando um *hash* de valores de campo, proporcionalmente aos pesos. O Open Vswitch faz um *hash* com os valores de vários campos incluindo endereços MAC, endereços IP, VLAN ID, tipo Ethernet, protocolo, para escolher um *bucket* da tabela de grupo. Os valores-hash resultantes são armazenados em cache pelo switch, para selecionar *bucket* uniformemente para cada fluxo. Portanto, vimos que o tráfego foi balanceado de forma equilibrada entre os *bucket*s, como foi observado pela taxa de pacotes transmitidos em cada interface de saída do switch.

Para trabalhos futuros, sugerimos um estudo de como o *hash* realizado no switch pode otimizar o balanceamento de carga. Também destaca-se uma interessante implementação apresentada no artigo "*SDProber: A Software Defined Prober for SDN"* (RAMANATAHN,2018). Neste artigo, os autores propuseram uma estratégia de medição proativa para detectar congestionamentos na rede definida por software, utilizando alguns dos principais conceitos e ferramentas apresentados neste projeto, que pode embasar uma proposta de estudo sobre QoS em redes definidas por software.

6 – Referências Bibliográficas

BOBBA,Rakesh; et al. **Software-Defined Networking Addresses Control System Requirements**. In: Sensible Cybersecurity for Power Systems: A Collection of Technical Papers Representing Modern Solutions, 2018. Disponível em: <https://cdn.selinc.com/assets/Literature/Publications/Technical%20Papers/6655_SoftwareDefined_RS_20140423_Web.pdf?v=20181015-212523>

CHIAO, Arthur. **OVS Deep Dive 0: Overview.** 2016. Disponível em: <https://arthurchiao.github.io/blog/ovs-deep-dive-0-overview/>

COMER, Douglas E. **Internetworking with TCP/IP - Principals, Protocols and Architecture**. Pearson, 6 ed.,2013.

COX, Jacob H. et al. **Advancing Software-Defined Networks: A Survey.** In: IEEE Access, 5, 2017. Disponível em < https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8066287>

FISCHETTI, Antonio; BODIREDDY, Bhanuprakash. **OVS-DPDK Datapath Classifier.** 2016. Disponível em: < https://software.intel.com/en-us/articles/ovs-dpdk-datapath-classifier>

GHOLAMI, M.; AKBARI, B. **Congestion control using openflow in software defined data center networks**. In: INTERNATIONAL ICIN CONFERENCE - INNOVATIONS IN CLOUDS, INTERNET AND NETWORKS,19,2016, Paris, p. 1–5. Disponível em: <http://dl.ifip.org/db/conf/icin/icin2016/1570224426.pdf >

GUEDES, Dorgival. et al. **Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento de pesquisas em Redes de Computadores**. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 30, 2012, Ouro Preto. Livro de minicursos. Disponível em: <https://www.researchgate.net/publication/260346033_Redes_Definidas_por_Software_uma_abordagem_sistemica_para_o_desenvolvimento_de_pesquisas_em_Redes_de_Computadores >

HERTIANA, S. N.; KURNIAWAN, A. **A joint approach to multipath routing and rate adaptation for congestion control in openflow software defined networ**k. In: Wireless and Telematics (ICWT), 2015. Disponível em <https://ieeexplore.ieee.org/document/7449209>

HUANG, Tim. **Path Computation Enhancement in SDN Networks**. Ontario. 2015 Disponível em: <https://digital.library.ryerson.ca/islandora/object/RULA%3A4465/datastream/OBJ/download/Path_computation_enhancement_in_SDN_networks.pdf

JOUET, S.;et al. **Otcp: Sdn-managed congestion control for data center networks**. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS),2016, p. 209–225.

KLEIS, E. G. **Desenvolvimento de uma aplicação para diferenciação de caminhos em redes definidas por software**. Universidade Federal Fluminense, Niterói, 2014.

KREUTZ, D. et al. **Software-defined networking: A comprehensive survey**. Proceedings of the IEEE, n. 103,2015, p. 14-76. Disponível em: <https://arxiv.org/pdf/1406.0440.pdf>

KUROSE James; ROSS, Keith. **Redes de Computadores e a Internet: Uma abordagem top-down**. Pearson, 7 ed., 2017.

LANTZ, Bob; O'CONNOR, Brian. **Mininet: An instant virtual network on your laptop**. Disponível em: <https://docplayer.net/40913267-Mininet-an-instant-virtual-network-on-your-laptop.html>

LARA, A.et al. **Network innovation using openflow: A survey.** In: Communications Surveys & Tutorials, IEEE, volume 16,2014 p. 493-512.

MA, Chloe. **SDN secrets of Amazon and Google**.2014. Disponível em: <https://www.infoworld.com/article/2608106/sdn/sdn-secrets-of-amazon-and-google.html>

ONF – Open Network Foundation. SDN architecture. 2014. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf>

ONF - Open Network Foundation . OpenFlow Switch Specification. 2015. Disponível em: <http://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>

PANTUZA,Gustavo. **O protocolo Openflow**. Disponível em: <https://blog.pantuza.com/artigos/o-protocolo-openflow>

RAMANATHAN, Sivaramakrishnan. et al. **SDProber: A Software Defined Prober for SDN.** In SOSR '18: ACMSIGCOMM Symposium on SDN Research, March 28–29, Los Angeles, CA, USA, 7 p. 2018. Disponível em: <https://dl.acm.org/citation.cfm?id=3185472>

RAO, Sridhar. **SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs.** 2014. Disponível em:< https://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/>

REZENDE,Pedro. et al. **Roteamento Multicaminhos em Redes Definidas por Software.** In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS,2016,Bahia. Disponível em < http://www.sbrc2016.ufba.br/downloads/SessoesTecnicas/152479.pdf>

RODRIGUES, Cristiane Pinto. **Avaliação de Balanceamento de Carga Web em Redes Definidas por Software**. Universidade Federal de Juiz de Fora, 2014. Disponível em : <http://monografias.nrc.ice.ufjf.br/tcc-web/downloadPdf?id=157>

RODRÍGUEZ, Fernando López. **Arquitetura e protótipo de uma rede SDN-OpenFlow para provedor de serviço**. Universidade de Brasília, 2014. Disponível em <http://repositorio.unb.br/bitstream/10482/16030/1/2014_FernandoLopezRodrigues.pdf>

SDxCentral. (2015). *SDxCentral SDN and NFV Market Size Report*. Disponível em: <https://www.sdxcentral.com/reports/sdn-nfv-market-size-forecast-report-2015/>

SONG, S. et al. **A congestion avoidance algorithm in SDN environment**. Information Networking (ICOIN), p. 420–423, jan 2016.

SENTHIL, Ganesh N. RANJANI S. **Dynamic Load Balancing using Software Defined Networks**. In: International Conference on Current Trends in Advanced Computing,2015. Disponível em: <https://pdfs.semanticscholar.org/4003/55f7f9632e6c2f33024c45788ed4ae279519.pdf>

WANG,R. et al. **OpenFlow-based server load balancing gone wild**. In: Proceedings of the USENIX conference on hot topics in management of internet, cloud, and enterprise networks and services, 11,2011, Berkeley. Disponível em:

<http://cial.csie.ncku.edu.tw/st2014/pdf/OpenFlowBased%20Server%20Load%20Balancing%20Gone%20Wild.pdf>

WENFENG Xia, Yonggang Wen, et al. **A Survey on Software-Defined Networking**.2015. Disponível em: < https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6834762 >

WILDAN, M. **Multipath Routing with Load Balancing using RYU OpenFlow Controller.**2018. Disponível em: <https://raw.githubusercontent.com/wildan2711/multipath/master/ryu_multipath.py>

YAO, H. et al**. A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks.** In: International Journal of Distributed Sensor Networks,11, 2015. Disponível em: <https://journals.sagepub.com/doi/abs/10.1155/2015/454159>