

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

GUSTAVO CARNEIRO DE SOUZA

IMPLEMENTAÇÃO DE UM CONTROLADOR ODL PARA SDN

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

GUSTAVO CARNEIRO DE SOUZA

IMPLEMENTAÇÃO DE UM CONTROLADOR ODL PARA SDN

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Augusto Foronda

PONTA GROSSA

2019



Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas



TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DE UM CONTROLADOR ODL PARA SDN

por

GUSTAVO CARNEIRO DE SOUZA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 12 de novembro de 2019 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Augusto Foronda
Orientador

Prof. MSc. Geraldo Ranthum
Membro titular

Prof. Dr. Richard Duarte Ribeiro
Membro titular

Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão
de Curso

Prof. Dr. André Pinz Borges
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Dedico este trabalho aos meus pais que nunca mediram esforços para me apoiar, que sempre acreditaram em mim, à vó Iracema (em memória) e à minha esposa pelo incentivo, pelo carinho e compreensão em minha ausência.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade de vida, não seria nada sem Ele, não teria chances de estar onde estou, oportunidade de estar entre as melhores universidades do Brasil, grato por tudo que tenho recebido Dele desde então.

Quero agradecer a minha família por todo o apoio, por toda a base que deram, pelos conselhos e exemplos.

À minha esposa, Camila, pelo incentivo, pelo carinho e compreensão em todos os momentos em que precisei para elaborar este trabalho.

Ao meu Professor e Orientador Prof. Dr. Augusto Foronda, por sua paciência, pelos seus conhecimentos, suas direções que me fizeram chegar até aqui, ajudando a se tornar real aquilo que um dia sempre almejei.

Aos meus amigos.

A secretaria do Curso e UTFPR em geral.

Peço desculpas caso tenha esquecido de alguém entre estas palavras, mas sabem a importância que cada um possui em minha vida e jornada.

Por fim, à todos que contribuíram diretamente ou indiretamente para meu sucesso neste trabalho.

RESUMO

SOUZA, Gustavo. **Implementação de um controlador ODL para SDN**. 2019. 36 f. Trabalho de Conclusão de Curso - Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Com o crescimento acelerado da tecnologia e das redes de computadores, surge a necessidade de expansão e/ou modernização destes ambientes de rede, seja para atualizar os dispositivos ou tornar os mesmos mais simples e práticos quanto ao seu gerenciamento. Contudo, o gerenciamento rígido de alguns dispositivos dificulta este controle. Nestes o código fonte é fechado, não sendo possível ser alterado pelo usuário, apenas pelo fornecedor. Com isso surgiram as Redes Definidas por *Software*, do inglês, *Software Defined Networking* (SDN), mantido pela *Linux Foundation*, um *software open source*, com uma ampla gama de desenvolvedores, membros e entidades colaborativas, que tem por objetivo trazer soluções para os problemas existentes. Possui uma arquitetura multi camadas, onde são separados os planos de dados, de administração e controle, possuindo em seus extremos APIs (*Application Programming Interface*) que conectam com roteadores, *switches* e com os mais variados protocolos, como por exemplo, o Openflow. Desta forma, este trabalho tem como objetivo explicar brevemente alguns controladores como ONOS (*Open Network Operating System*), Floodlight e RYU, demonstrar a instalação e a implementação de um controlador *OpenDaylight* (ODL) virtualizado no Virtualbox e um ambiente de comutadores via Mininet para criação da topologia de rede, permitindo os testes em homologação para posteriormente o uso em produção. Por fim mostrar algumas formas de gerenciamento da rede, o que é possível fazer e onde encontrar tanto pelo controlador quanto pelo Mininet, exemplificando a importância de possuir toda uma rede em um único lugar, centralizado, deixando em evidência uma das principais características do controlador SDN.

Palavras-chave: SDN. ODL. Openflow.

ABSTRACT

SOUZA, Gustavo Carneiro de. **ODL Controller's Implementation for SDN**. 2019. 36 p. Work of Conclusion Course - Graduation in Analysis and Systems Development - Federal Technology University - Paraná. Ponta Grossa, 2019.

By the exponential growth of technology and computer networks, there is a need to expand and / or modernize these network environments, either to upgrade devices or make them simpler and more practical to manage. However, the stuck management of some devices makes this control difficult, where the source code is closed, and it is not possible to change at the user level, only by the supplier. This way, the Software Defined Networking (SDN), hosted by the Linux Foundation, an open source software, with a broad academy of developers, members and collaborative entities, which aims to bring solutions to these problems. It has a multi-layered architecture, where data, administration and control plans are basically separated, having at their extremes APIs that connect with routers, switches and various protocols, such as the well-known Openflow. Thus, this paper aims to briefly explain some controllers such as Open Network Operating System (ONOS), Floodlight and RYU, to demonstrate the installation and implementation of a virtualized OpenDaylight (ODL) controller in Virtualbox and a Mininet to give the possibility of creating a topology with the switch's environment, allowing for homologation testing for later production use. Finally, show some forms of network management, what you can do and where to find both as the controller as Mininet, exemplifying the importance of having a whole network in one place, centralized, highlighting one of the main features of the SDN controller.

Keywords: SDN. ODL. Openflow.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura Resumida SDN	15
Figura 2 - Transição da Arquitetura Tradicional para a Arquitetura SDN	16
Figura 3 - Alguns Controladores SDN	17
Figura 4 - Arquitetura Openflow	18
Figura 5 - Exemplificação Tabela de Fluxo	19
Figura 6 - Transição da NFV	21
Figura 7 - Arquitetura ONOS	22
Figura 8 - Exemplo Arquitetura FloodLight	23
Figura 9 - RYU Framework	24
Figura 10 - Arquitetura OpenDayLight	26
Figura 11 - Visão da Rede da Arquitetura ODL	26
Figura 12 - Instalação do Controlador ODL e Mininet	28
Figura 13 - Inicializando o controlador ODL	29
Figura 14 - Instalação das funcionalidades ODL	30
Figura 15 - Interface gráfica ODL sem topologia	31
Figura 16 - Conexão ao Mininet via SSH	31
Figura 17 - Criação da Topologia Ramificada	32
Figura 18 - Dispositivos adicionados à rede Mininet	32
Figura 19 - Ping entre hosts h1 e h9 na Topologia Ramificada	33
Figura 20 - Ping entre hosts h7 e h15 na Topologia Ramificada	33
Figura 21 - Criação da Topologia Linear	34
Figura 22 - Topologia Linear	34
Figura 23 - Ping entre hosts h10 e h1	35
Figura 24 - Inativação do Switch	35
Figura 25 - Lista de Switches Ativos	36
Figura 26 - Node Connectors	36

LISTA DE SIGLAS

ODL	<i>Open Day Light</i>
API	<i>Application Programming Interface</i>
SDN	<i>Software-defined Networking</i>
ONOS	<i>Open Network Operating System</i>
ONF	<i>Open Networking Foundation</i>
GPL	<i>General Public License</i>
NFV	<i>Network Functions Virtualization</i>
SB	<i>Southbound</i>
MD-SAL	<i>Model-Driven Service Abstraction Layer</i>

SUMÁRIO

1 INTRODUÇÃO	11
1.1 OBJETIVO	12
1.1.1 Objetivo Geral	12
1.1.2 Objetivos Específicos	12
1.2 JUSTIFICATIVA	12
1.3 ORGANIZAÇÃO DO TRABALHO	12
2 REFERENCIAL TEÓRICO	14
2.1 REDES DEFINIDAS POR <i>SOFTWARE</i>	14
2.2 CONTROLADOR SDN	16
2.3 <i>OPENFLOW</i>	17
2.4 <i>OPENFLOW</i> E SEUS COMPONENTES	18
2.4.1 Tabela de Fluxo	19
2.4.2 Controlador	19
2.5 NFV	20
3 CONTROLADORES SDN	22
3.1 ONOS	22
3.2 FLOODLIGHT	23
3.3 RYU	23
3.4 OPENDAYLIGHT	24
4 IMPLEMENTAÇÃO DO CONTROLADOR ODL	27
4.1 <i>SOFTWARES</i> UTILIZADOS	27
4.1.1 Virtualbox	27
4.1.2 Mininet	28
4.2 INSTALAÇÃO ODL E MININET	28
4.3 FUNCIONAMENTO DO ODL E MININET	30
5 CONCLUSÃO	37
5.1 TRABALHOS FUTUROS	37
REFERÊNCIAS	38

1 INTRODUÇÃO

Projetar e gerenciar as redes tem se tornado menos complexo com o passar dos anos com a ajuda das Redes Definidas por *Software* (SDN). Há a impressão de que esta tecnologia apareceu repentinamente, porém, ela faz parte de uma longa história, onde vem tentando fazer as redes de computadores mais programáveis (FEAMSTER et al, 2013).

Atualmente, organizações estão aumentando significativamente o uso de ambientes computacionais avançados para atingir suas necessidades, incluindo redes baseadas em nuvem, *desktops* e servidores virtualizados, dispositivos de armazenamento de dados, tecnologias estas que requerem mais recursos computacionais (KIRKPATRICK, 2013).

Redes de computadores são complexas e difíceis de gerenciar. Envolve vários tipos de equipamentos, de roteadores e *switches* à aplicação de Internet (*middleboxes*) como *firewalls*, balanceadores de carga de servidores, entre outros. Administradores de redes tipicamente configuram dispositivos de rede individualmente usando interfaces de configuração que podem variar entre fornecedores e até mesmo variar entre produtos deste mesmo fornecedor (FEAMSTER et al, 2013).

A SDN é caracterizada pela existência de um sistema de controle, onde existe um controlador (*software*), que comanda o mecanismo de transferência de dados dos elementos de comutação da rede por uma interface de programação bem definida. Sendo mais direto, os elementos de comutação exportam uma interface de programação que permite ao *software* inspecionar, definir e alterar entradas da tabela de roteamento do comutador como ocorre com comutadores Openflow (GUEDES et al, 2012).

Existem alguns controladores comerciais, como: o *OpenDayLight*, *ONOS* (*Open Network Operating System*), *RYU*, *POX*, *FloodLight*, entre outros. Alguns destes serão explicados mais detalhadamente no decorrer do trabalho, assim como a implementação do ODL que é o foco principal deste trabalho.

1.1 OBJETIVO

1.1.1 Objetivo Geral

Este trabalho tem como objetivo implementar e analisar um controlador ODL para rede SDN.

1.1.2 Objetivos Específicos

- Estudar os conceitos e definições das Redes Definidas por *Software* (SDN);
- Definir o ambiente de instalação e simulação da topologia de testes;
- Implementar o ODL e mininet;
- Realizar testes para mostrar o funcionamento da topologia.

1.2 JUSTIFICATIVA

Devido ao grande crescimento de dispositivos conectados as redes (roteadores, *switches*, etc), é de extrema importância pensar em como seria possível configurar estes dispositivos de forma mais autônoma e flexível. O objetivo é que haja a possibilidade de homologar em um ambiente virtual e colocar em produção quando configurado e completamente funcional.

Encontra-se limitações nos roteadores e *switches* criados por seus fornecedores, onde não é possível visualizar, muito menos alterar as configurações entregues por estes fornecedores.

As Redes Definidas por *Software* trazem uma solução para este problema e este trabalho mostra como implementar uma solução SDN usando o controlador ODL.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho será organizado da seguinte maneira. No capítulo 2 será explicada a estrutura das Redes Definidas por *Software*, assim como o *Openflow* e seus

componentes. O capítulo 3 descreve os principais controladores e suas características. No capítulo 4 detalha a implementação do controlador ODL. No capítulo 5 é feita a conclusão do trabalho.

2 REFERENCIAL TEÓRICO

2.1 REDES DEFINIDAS POR SOFTWARE

O termo SDN tem se tornado mais conhecido nos últimos anos, mas o conceito já vem sendo estudado desde 1996 por vários pesquisadores, impulsionados pelo desejo de fornecer um gerenciamento de dados controlado pelo usuário nos nós de rede. Muitos grupos de empresas trabalharam na implementação do SDN, entre eles, *Ethane* (2007) e *Openflow* (2008). Estes trouxeram para mais perto a implementação do SDN a realidade das redes (SEZER et al, 2013).

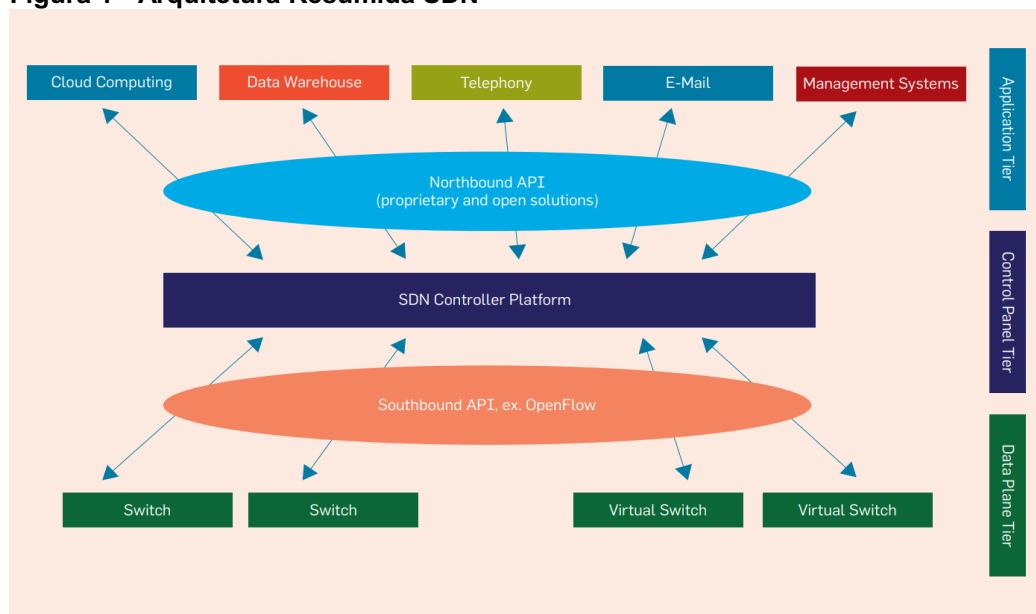
A definição sobre SDN dada pela *Open Networking Foundation* (ONF, 2014), é descrita como uma arquitetura, onde os planos de controle e de dados são desacoplados, a inteligência da rede e estado são logicamente centralizados.

A ONF, que regulariza e padroniza a SDN, define que a arquitetura SDN possui três camadas distintas que são acessadas através das APIs:

- Camada de Aplicação: Consiste na aplicação do usuário final que consome os serviços de comunicação SDN. Entre a camada de aplicação e a camada de controle a comunicação é feita através das APIs;
- Camada de Controle: Fornece as funcionalidades de controle logicamente centralizado que supervisiona o comportamento do encaminhamento de dados através de uma interface aberta;
- Camada de Infraestrutura: Consiste dos elementos de rede e dispositivos que fornecem a troca e o encaminhamento de pacotes.

A Figura 1 mostra resumidamente como a arquitetura funciona com suas respectivas camadas, as APIs Norte e Sul. Pode-se ver a Camada de Aplicação como *Application Tier*, a Camada de Controle como *Control Panel Tier* e a Camada de Infraestrutura como *Data Plane Tier*.

Figura 1 - Arquitetura Resumida SDN



Fonte: (KIRKPATRICK, 2013)

Entre as camadas da arquitetura SDN existem as APIs e por meio destas é feita a comunicação:

- **API Sul (Southbound API):** Representa a interface de comunicação entre o plano de controle e o plano de dados;
- **API Norte (Northbound API):** SDN permite a troca de informação com aplicações. Fica entre as camadas de controle e aplicação.

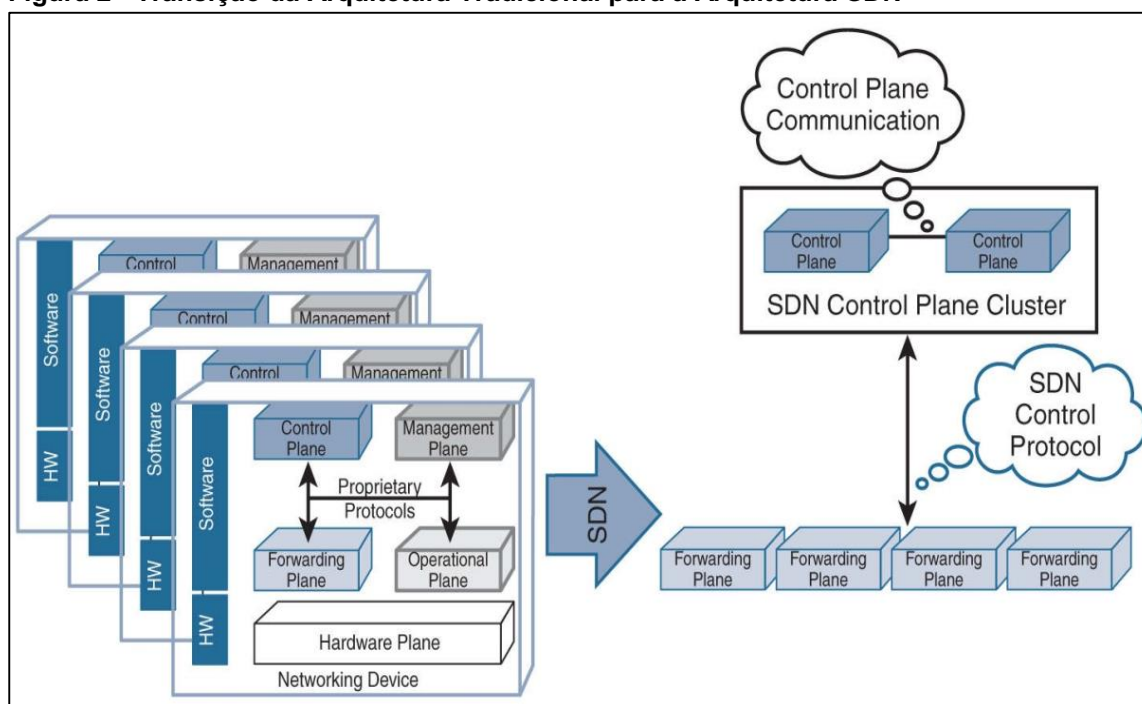
Uma característica muito importante que pode ser citada é a simplificação dos dispositivos que são controlados por um sistema centralizado de gerenciamento e *software* de controle. Ao invés de milhares de linhas de *software* com um plano de controle complicado executando no dispositivo, fazendo com que o dispositivo trabalhe de forma autônoma, essa configuração é retirada, e assim, colocada em um controlador centralizado (GÖRANSSON;BLACK, 2014).

O controlador gerencia a rede usando regras de alto nível e este por sua vez, fornece instruções para os dispositivos simplificados, ajudando estes a tomarem as decisões corretas quanto aos pacotes recebidos (GÖRANSSON;BLACK, 2014)

Portanto, o objetivo do SDN é desacoplar o plano de controle do plano de encaminhamento de dados, mas não obriga o plano de controle centralizado seja confinado a um único nó. Justamente pela escalabilidade e alta disponibilidade, o plano de controle pode ser expandido horizontalmente, formando um *cluster* de plano

de controle. A Figura 2 mostra os conceitos do SDN e as mudanças que isso traz comparado com a arquitetura de rede tradicional. Como o foco é nos planos de controle e encaminhamento de dados a figura não enfatiza a interação dos planos de *hardware* ou o plano operacional e de gerenciamento (CHAYAPATHI et al, 2016).

Figura 2 - Transição da Arquitetura Tradicional para a Arquitetura SDN



Fonte: (CHAYAPATHI et al., 2016)

2.2 CONTROLADOR SDN

O controlador SDN é um dispositivo independente que age como o plano de controle SDN e comunica as decisões para os dispositivos de rede. Ele também recupera informação dos dispositivos para tomar decisões conscientes para o plano de controle. Para sua comunicação com os dispositivos usa-se protocolos, que são os protocolos de controle SDN (CHAYAPATHI et al., 2016).

A Figura 3 mostra os principais tipos de controladores e suas características.

Figura 3 - Alguns Controladores SDN

Nome	Linguagem	Plataforma	Característica
NOX	C++, Python	Linux	Controlador de referência OpenFlow
POX	C++, Python	Windows, Mac, Linux	Evolução do NOX
Maestro	Java	Windows, Mac, Linux	Explora o paralelismo
Beacon	Java	Windows, Mac, Linux, Android	Multiplataforma, Multithreaded
Floodlight	Java	Windows, Mac, Linux	Pode-se integrar a redes não OpenFlow
Frenetic	Funcional/Declarativa Própria	Linux	Programa a rede como um todo
Onix	C++, Python, Java	Windows, Mac, Linux	Gerência redes de grande porte
SNAC	C++	Linux	Monitoramento de redes OpenFlow; Interface Web
Trema	C, Ruby	Linux	Script; Emulador

Fonte: (COSTA, 2013)

2.3 OPENFLOW

Openflow é um protocolo de comunicação entre o controlador e os dispositivos de rede e gerencia todo o ambiente de encaminhamento de dados destes dispositivos (SILVA;FARIA, 2017).

Podendo ser aplicado tanto em produção quanto em homologação devido a um mecanismo que é ativado em todos os computadores. Pesquisadores podem experimentar ou reprogramar sem ter que se preocupar com o fluxo de produção do ambiente (COSTA, 2013).

“Outro objetivo da proposta do *Openflow* é permitir que os fabricantes de *hardware* possam inserir as funcionalidades do *Openflow* nos seus computadores sem a necessidade de expor o projeto desses equipamentos. Um requisito importante nessa abordagem, já que a infraestrutura será modificada, é que tais equipamentos devem possuir um custo baixo e um desempenho semelhante aos já utilizados na rede de produção, de maneira que se torne viável para os administradores de rede a troca desses equipamentos pelos já utilizados” (COSTA, 2013).

Segundo (COSTA, 2013), o projeto *Openflow* define:

- Ser e dar suporte a pesquisas científicas;
- Capacitação da implementação (baixo x alto desempenho);

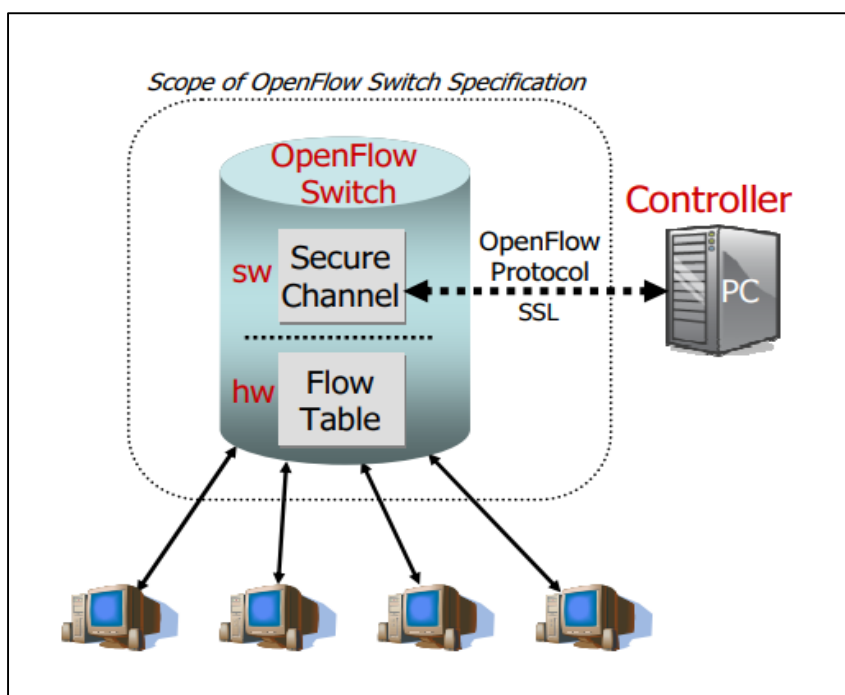
- Dar garantia do tráfego de produção e homologação quanto ao seu isolamento;
- Dar garantia da não necessidade de exposição do projeto dos fabricantes em suas respectivas plataformas.

A Arquitetura *Openflow* resume-se em (SILVA;FARIA, 2017):

- Um controlador para gerenciar o plano de controle;
- Tabela de Fluxo para armazenamento de dados de cada fluxo;
- Um canal seguro, onde há conexão entre controlador e *hardware* responsável pelo encaminhamento de dados/pacotes;
- Protocolo *Openflow* que fará a troca de mensagens.

A Figura 4 mostra os 4 elementos acima da arquitetura *Openflow*.

Figura 4 - Arquitetura *Openflow*



Fonte: (MCKEOWN et al, 2008)

2.4 OPENFLOWE SEUS COMPONENTES

Nos capítulos abaixo serão explicados os componentes do *Openflow*.

2.4.1 Tabela de Fluxo

A tabela de fluxo está no centro da definição de um comutador *Openflow*. Possui entradas de fluxo, e estas consistem em contadores, regras e ações de cabeçalho. Este campo de cabeçalho é responsável por identificar se, os pacotes que estão chegando, combinam com esta entrada, caso seja classificado como igual este pacote pertence a este fluxo. Os contadores servem como estatística, como por exemplo, quantos pacotes foram encaminhados ou recebidos por tal fluxo de entrada (GÖRANSSON; BLACK, 2014).

Conforme mostra a Figura 5, possui os fluxos de entrada, os cabeçalhos, os contadores e ações.

Figura 5 - Exemplificação Tabela de Fluxo

Flow Entry 0		Flow Entry 1			Flow Entry F			Flow Entry M	
Header Fields	Inport 12 192.32.10.1, Port 1012	Header Fields	Inport * 209.*.*.*, Port *		Header Fields	Inport 2 192.32.20.1, Port 995		Header Fields	Inport 2 192.32.30.1, Port 995
Counters	val	Counters	val	■■■	Counters	val	■■■	Counters	val
Actions	val	Actions	val		Actions	val		Actions	val

Fonte: (GÖRANSSON; BLACK, 2014)

2.4.2 Controlador

Como explicado por (MCKEOWN et al, 2008), a ideia de que o controlador é responsável por adicionar e remover os fluxos de entrada da tabela de fluxos. “É uma camada de abstração da infraestrutura física que tem como objetivo facilitar o desenvolvimento de aplicações e serviços que gerenciam as entradas de fluxo de rede” (COSTA, 2013, p. 42).

Responsável pela organização do ambiente, com características de um SO, ele faz a gerência e todo o controle necessário às redes (ROTHENBERG et al, 2010).

2.5 NFV

O termo NFV (*Network Functions Virtualization*) é construído do conceito de virtualização de servidor, mas vai muito além deste conceito de servidores, incluindo dispositivos de redes. A NFV permite gerenciar, provisionar, monitorar e implantar esses dispositivos virtualizados (CHAYAPATHI et al, 2016).

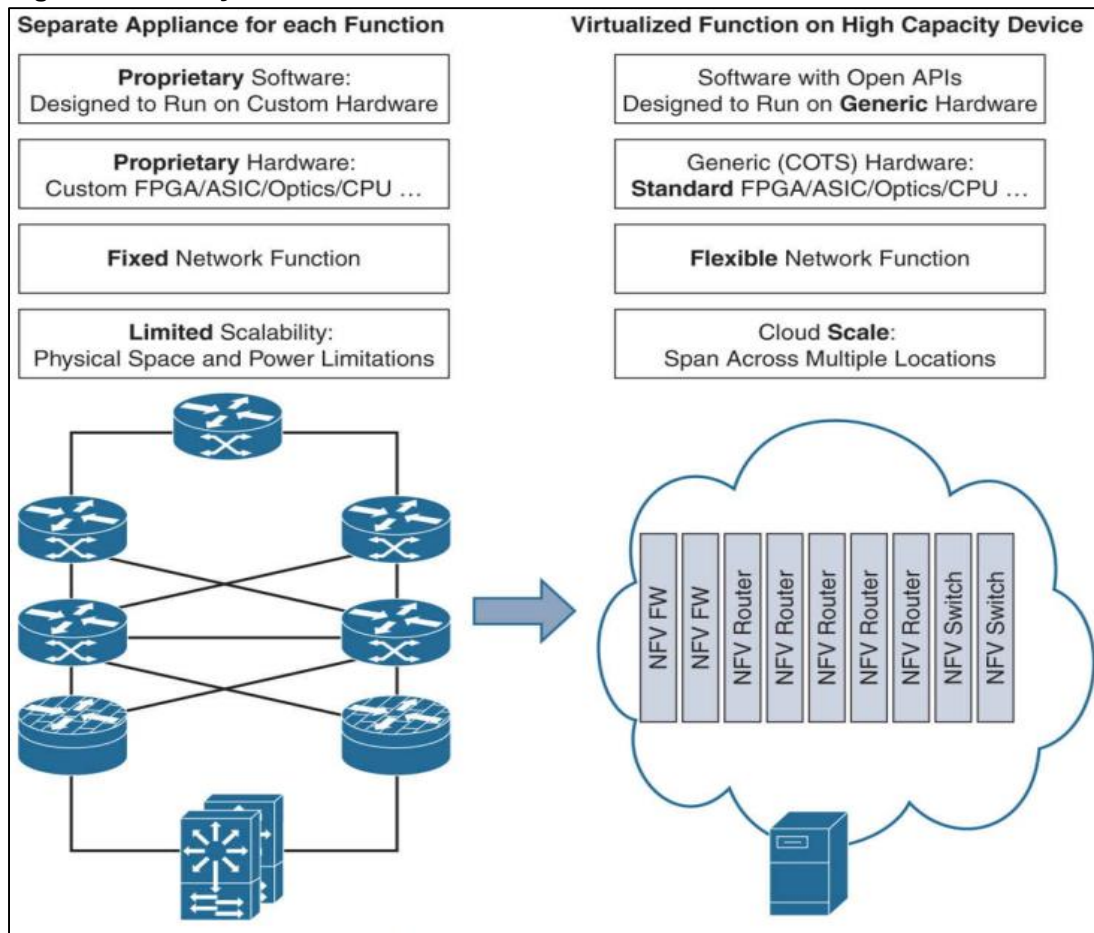
NFV é usado como um termo geral para referenciar todo o ecossistema no que compreende os dispositivos de redes virtuais, ferramentas de gerenciamento e infraestrutura que integra partes de *software* com *hardware*.

Portanto, o NFV é definido como um método e tecnologia que permite você trocar os dispositivos de rede que executam funções específicas de rede por um programa ou mais executando as mesmas funções de rede enquanto executam em *hardware* de computadores genéricos. Pode-se citar como exemplo, substituir uma aplicação de *firewall* (física) com uma máquina virtual. Essa máquina virtual fornece as funções do *firewall*, roda o mesmo sistema operacional, mas em um *hardware* genérico (CHAYAPATHI et al, 2016).

Com NFV, as funções de rede podem ser implementadas em um *hardware* genérico que fornece alguns pré-requisitos como recursos básicos para processamento, armazenamento e transmissão de dados (CHAYAPATHI et al, 2016).

A Figura 6 mostra a transição da NFV.

Figura 6 - Transição da NFV



Fonte: (CHAYAPATHI et al, 2016)

3 CONTROLADORES SDN

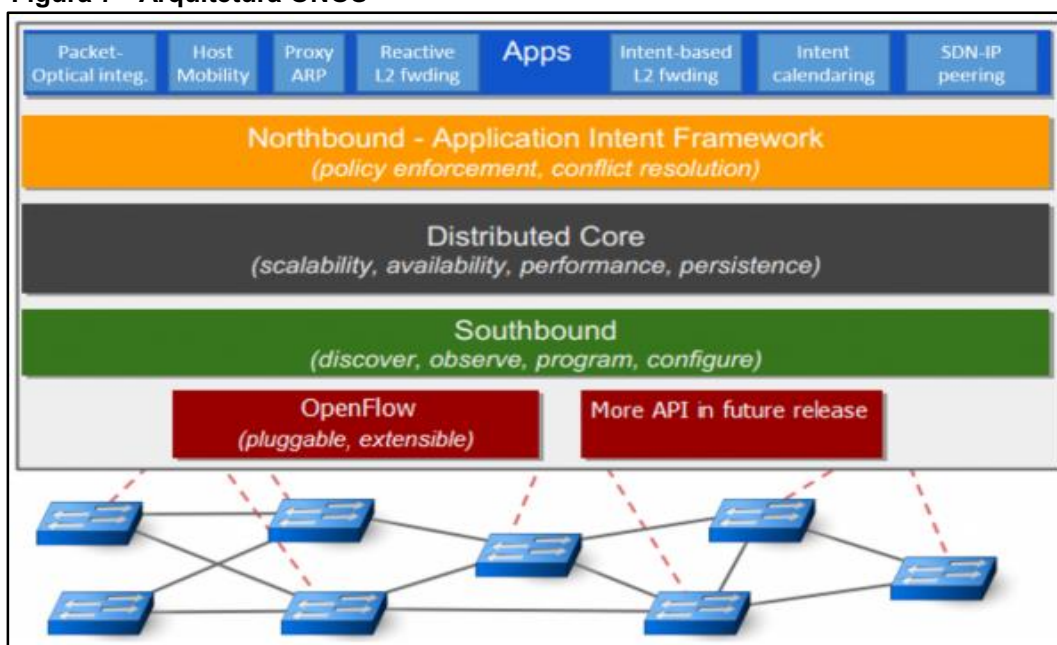
Esta seção descreve alguns controladores, entre eles o ONOS, RYU e FloodLight. O controlador usado neste trabalho é o ODL, que explicado após a descrição dos outros controladores.

3.1 ONOS

Disponibilizado em código aberto, fornece o plano de controle para SDN, gerenciando os componentes de rede, tais como *switches* e roteadores. Fornece também serviço de comunicação através de programas e módulos (ONOS, 2019).

Possuindo grande capacidade de escalabilidade, alto desempenho, devido ao seu núcleo distribuído e responsável por manter a rede e comunicação através de APIs. Possui licença do Apache 2.0 e uma vasta comunidade de desenvolvedores (ONF, 2019). A Figura 7 mostra a arquitetura do controlador ONOS e suas características.

Figura 7 - Arquitetura ONOS



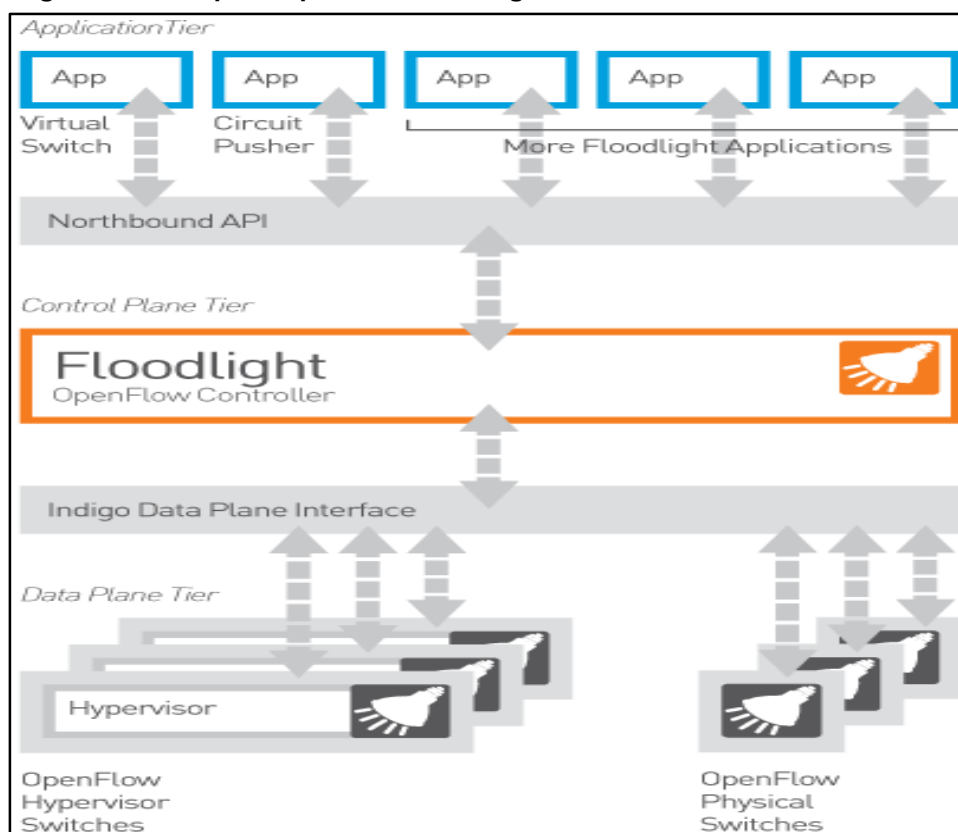
Fonte: (ONF, 2019).

3.2 FLOODLIGHT

Tem toda sua base em Java, atualmente gerenciado pela ONF, possuindo uma comunidade aberta de desenvolvimento, Apache licenciado permitindo ser usado por quase toda e qualquer finalidade. É de fácil uso com a documentação à disposição, e foi criado para trabalhar com uma grande quantidade de *switches* físicos e virtuais, onde conversam pelo protocolo *Openflow* (FLOODLIGHT, 2019).

Figura 8 exemplifica a arquitetura do controlador FloodLight.

Figura 8 - Exemplo Arquitetura FloodLight



Fonte: (FLOODLIGHT, 2019).

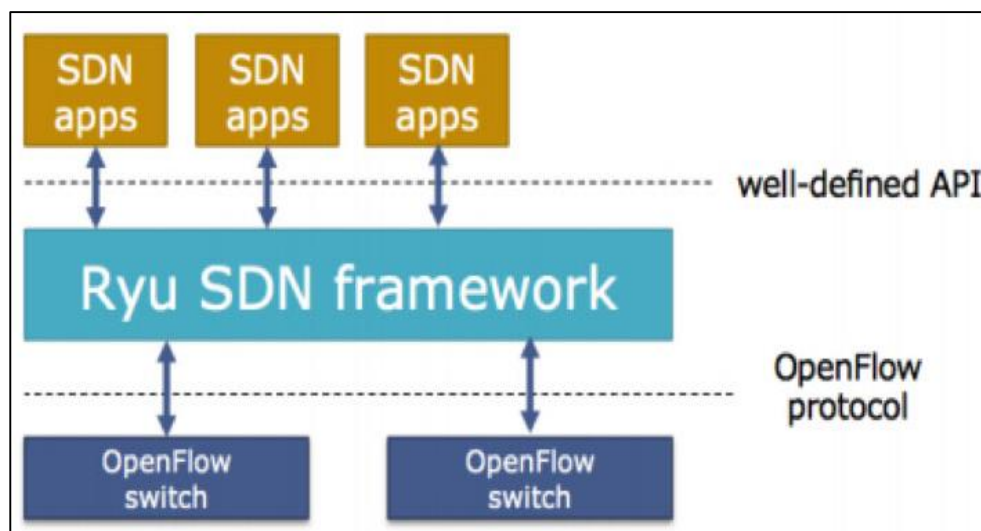
3.3 RYU

O controlador RYU é um *framework* SDN baseado em componentes, fornece API para tornar mais fácil a criação e o gerenciamento de aplicações de controle.

Assim como os outros controladores suporta vários tipos de protocolos, como por exemplo, *Openflow*, *Netconf*, entre outros. Licenciado também pelo apache 2.0 (SDX, 2016).

De forma compacta a Figura 9 mostra a arquitetura RYU.

Figura 9 - RYU Framework



Fonte (SDX, 2016).

3.4 OPENDAYLIGHT

Mantido pelo Linux Foundation o OpenDaylight é um projeto de código aberto que possui uma comunidade composta por desenvolvedores e membros para colaborar com o principal objetivo do projeto, que seria dar suporte e mantê-lo (BADOTRA et al, 2017).

Pode ser considerado como o componente central de uma arquitetura SDN, pelo fato de ser um *software* de código aberto, permite aos usuários minimizar a complexidade operacional e ainda, estendendo a vida útil da infraestrutura existente (BADOTRA et al, 2017).

É um controlador formado por uma arquitetura altamente disponível, modular, escalável e multiprotocolo, criado para atender as implantações SDN em redes modernas e heterogêneas dos mais variados fornecedores, controlando os elementos da rede (comutadores). Orientado a modelos, possui a possibilidade de escrita de apps, que funcionam facilmente em uma variedade de protocolos da camada inferior.

Vale ressaltar que a principal característica deste controlador é a camada de abstração de serviço orientada a modelos MD-SAL (*Model-Driven Service Abstraction Layer*) (OPENDAYLIGHT, 2019).

O MD-SAL conecta os *plugins* de protocolos e módulos *Service Network Function*. O *API-Driven SAL* (AD-SAL) está sendo migrado aos poucos para o MD-SAL. O MD-SAL oferece a unificação entre os APIs *Northbound* e *Southbound*, incluindo a estrutura de dados usada em vários componentes do controlador (EFREMOVA; ANDRUSHKO, 2015).

Conforme citado anteriormente, para Badotra e Singh (apud MEDVED, 2014), existem duas abordagens diferentes quanto ao SAL, AD-SAL e MD-SAL, definindo assim algumas de suas características abaixo.

AD-SAL:

- Pode ser usado tanto em *Southbound* ou *Northbound plugins*;
- Sem estado;
- As aplicações são programadas dentro do controlador como os pacotes

OSGi (*Open Service Gateway Interface*);

- A programação do fluxo é reativa, recebendo eventos da rede.

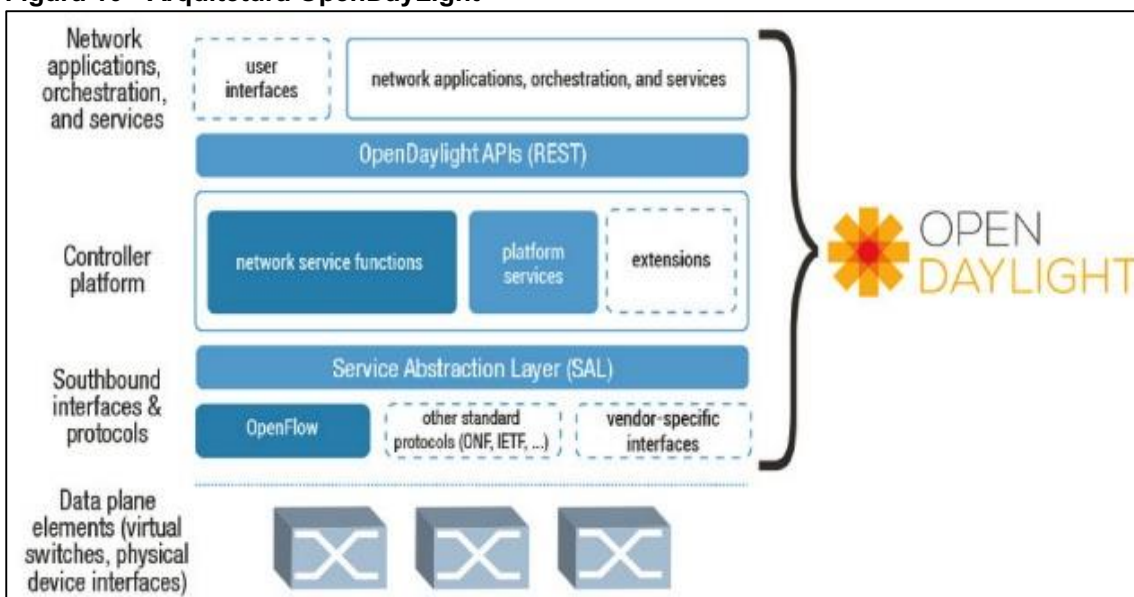
MD-SAL:

- Tem em comum o REST API para todos os módulos;
- Pode armazenar dados para os modelos em APIs permanentes ou voláteis;
- Suporta qualquer dispositivo ou modelos de serviço;
- A aplicação é programada fora do controlador.

A Figura 10 mostra a arquitetura OpenDayLight, onde Badotra (apud ONF, 2012; MEDVED, 2014) explica arquitetura como multicamadas, onde a camada principal é a plataforma do controlador, porque o controlador reside nela e age como um cérebro à rede, pois gerencia o fluxo do tráfego dos *switches* usando as tabelas de fluxo.

O controlador pode ser rodado em qualquer sistema operacional com um JVM instalado, múltiplos protocolos, como por exemplo, *Openflow 1.0*, *Openflow 1.3*, BGP-LS, entre outros na camada *Southbound* (BADOTRA et al, 2017).

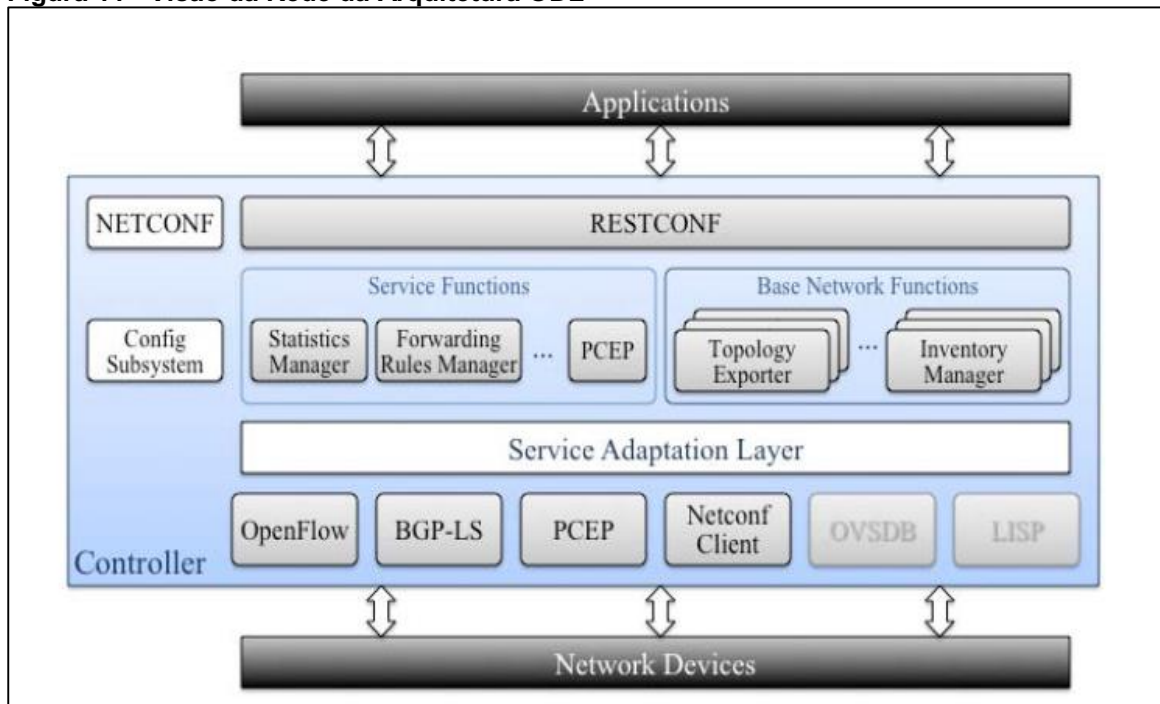
Figura 10 - Arquitetura OpenDayLight



Fonte: (BADOTRA et al., 2017)

A Figura 11 mostra a visão da rede na arquitetura do controlador ODL, uma vez que a arquitetura em camadas permite que o controlador suporte vários protocolos SB (*Southbound*), por intermédio dos *plugins* e fornece um conjunto uniforme de serviços e APIs para as aplicações (BADOTRA et al, 2014).

Figura 11 - Visão da Rede da Arquitetura ODL

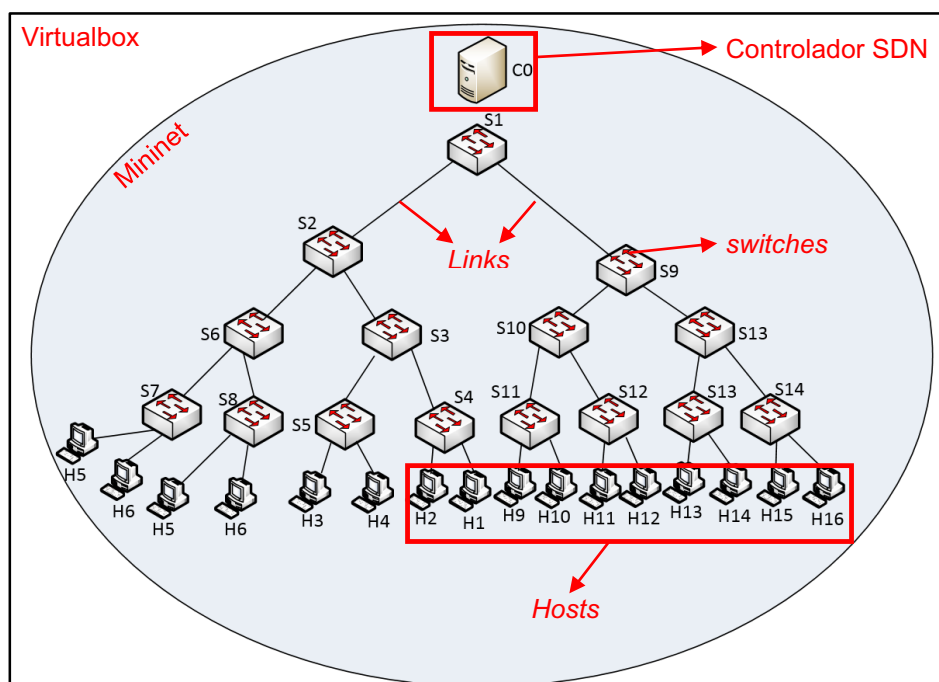


Fonte: (BADOTRA, 2017)

4 IMPLEMENTAÇÃO DO CONTROLADOR ODL

Esta seção descreve a implementação do controlador ODL juntamente com o Mininet. A Figura 12 explica em forma compacta a junção entre os sistemas:

Figura 12 - SDN na prática



Fonte: Adaptado (CESAR, 2016)

4.1 SOFTWARES UTILIZADOS

Foram utilizados os seguintes *softwares* para demonstrar sua utilização e implementação.

4.1.1 Virtualbox

Ferramenta para virtualização tanto empresarial quanto para uso doméstico. Suas características ricas e de código aberto sob os termos do GNU *General Public License (GPL)*. Roda em uma diversidade de sistemas operacionais, contendo versões para Windows, Linux, Macintosh e Solaris, e suportando entre estes sistemas operacionais suas respectivas versões (VIRTUALBOX, 2019).

4.1.2 Mininet

Para resolver a dificuldade de implementar novos conceitos em redes corporativas e/ou indústrias, o Mininet permite criar redes virtuais, rodando com um kernel real, *switch* e códigos de aplicação em uma única máquina (VM, em nuvem ou nativa), desta maneira muito próximo da realidade, fornecendo a oportunidade de testes em homologação (MININET, 2018).

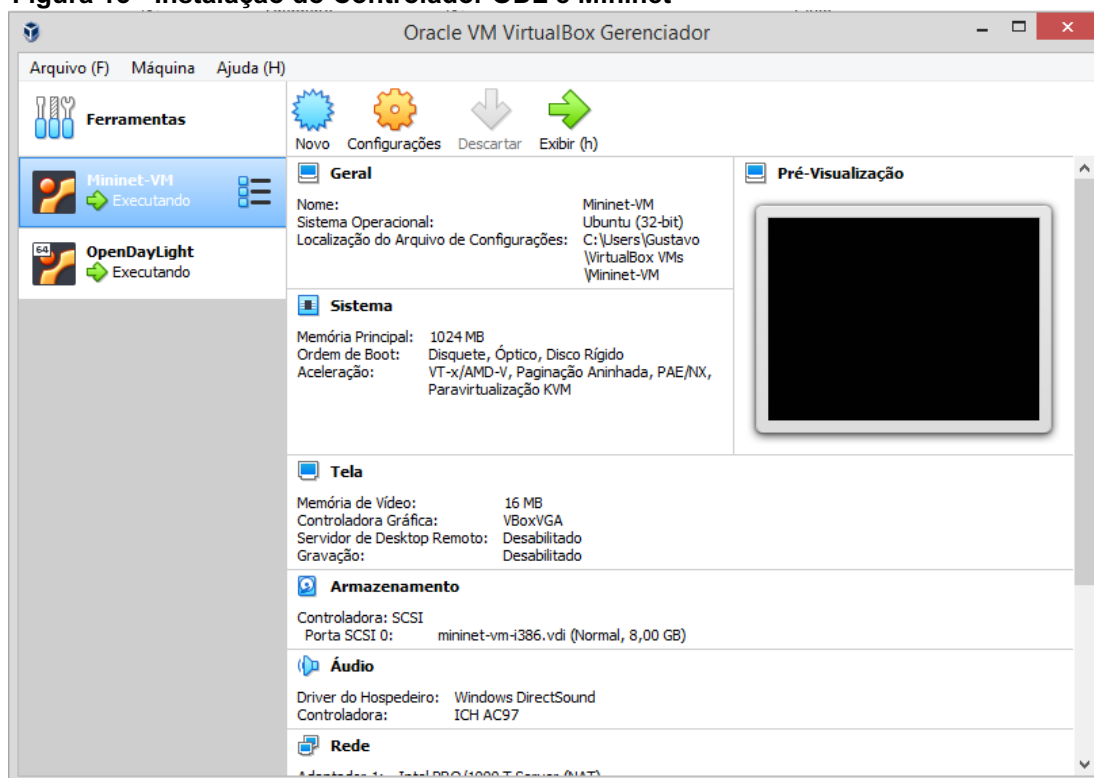
Ferramenta de código aberto, possui uma grande comunidade que compartilha soluções, documentação e wiki, permitindo uma contribuição compartilhada para a ferramenta (MININET, 2018).

4.2 INSTALAÇÃO ODL E MININET

Foi utilizada uma versão do Linux Ubuntu 14.04 LTS para instalação do controlador ODL. A imagem deste é fornecida pela comunidade ODL, assim como a imagem do Mininet. Ambos foram virtualizados utilizando o Virtualbox.

A Figura 13 mostra a instalação de ambos dentro do VirtualBox.

Figura 13 - Instalação do Controlador ODL e Mininet



Fonte: (VIRTUALBOX, 2019)

- odl-restconf: API que retorna XML de acordo com a requisição.

A Figura 15 a seguir exemplifica a instalação do ODL.

Figura 15 - Instalação das funcionalidades ODL

```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown
OpenDaylight.

opendaylight-user@root>feature:install odl-l2switch-switch-ui
```

Fonte: Autoria Própria

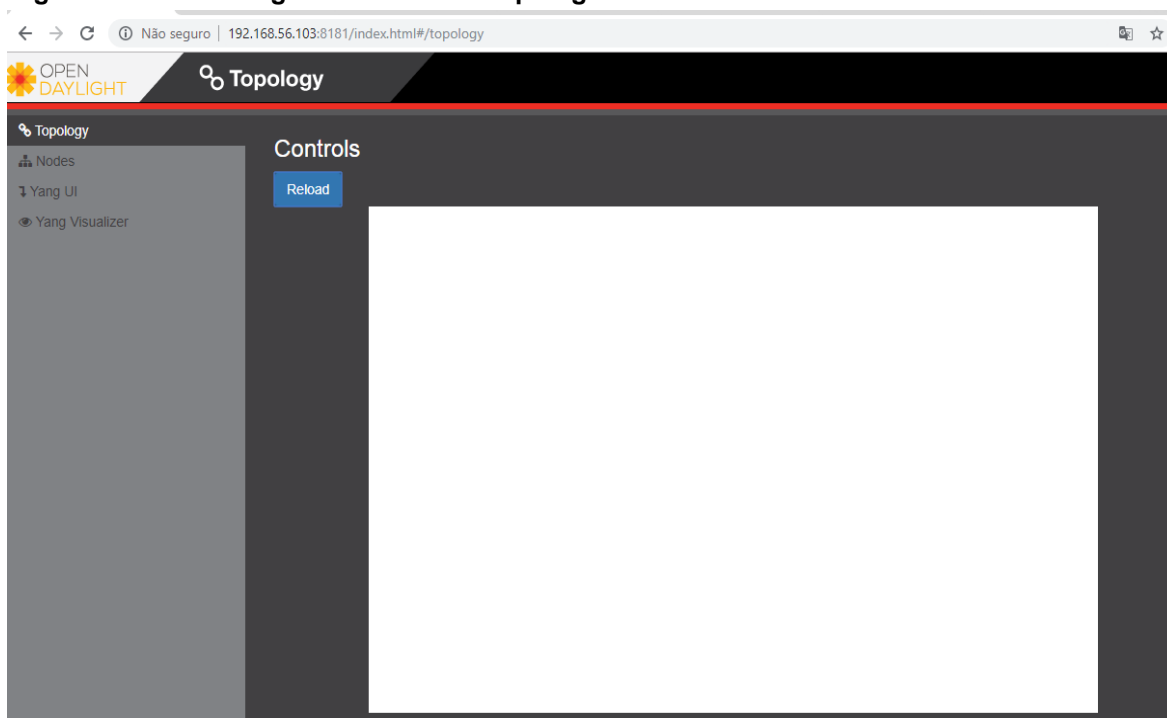
Switches tradicionais operam na camada 2 do modelo OSI, onde os pacotes são encaminhados para uma porta específica de um switch baseado no endereço MAC de destino (CISCO, 2018).

Quando estes pacotes chegam no *switch*, este verifica o endereço do MAC destino, se for conhecido, envia o pacote para a porta de saída (CISCOPRESS, 2003).

4.3 FUNCIONAMENTO DO ODL E MININET

Nesta seção, será detalhada a criação das topologias, criadas a partir de um API em python. Com este API é possível adicionar *switches* e *hosts* com apenas algumas linhas.

Primeiramente, pode-se notar na interface gráfica de que não há nenhuma topologia criada, conforme mostra a Figura 16.

Figura 16 - Interface gráfica ODL sem topologia

Fonte: Autoria Própria

Para adicionar as topologias, já com a VM Mininet ligada, conecta-se via SSH pelo terminal do Ubuntu e desta maneira existirá um terminal específico para usufruir das funcionalidades do Mininet. A Figura 17 exemplifica esta conexão.

Figura 17 - Conexão ao Mininet via SSH

```
opendaylight@opendaylight-VirtualBox:~$ ssh -X mininet@192.168.56.104
mininet@192.168.56.104's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)
```

Fonte: Autoria Própria.

Após a conexão e o controlador adicionado a rede, é criada as topologias via Mininet. Primeiramente é criada uma rede com 15 *switches*, 16 *hosts* e *links* interligando-os, conforme a Figura 18 abaixo:

Figura 18 - Criação da Topologia Ramificada

```

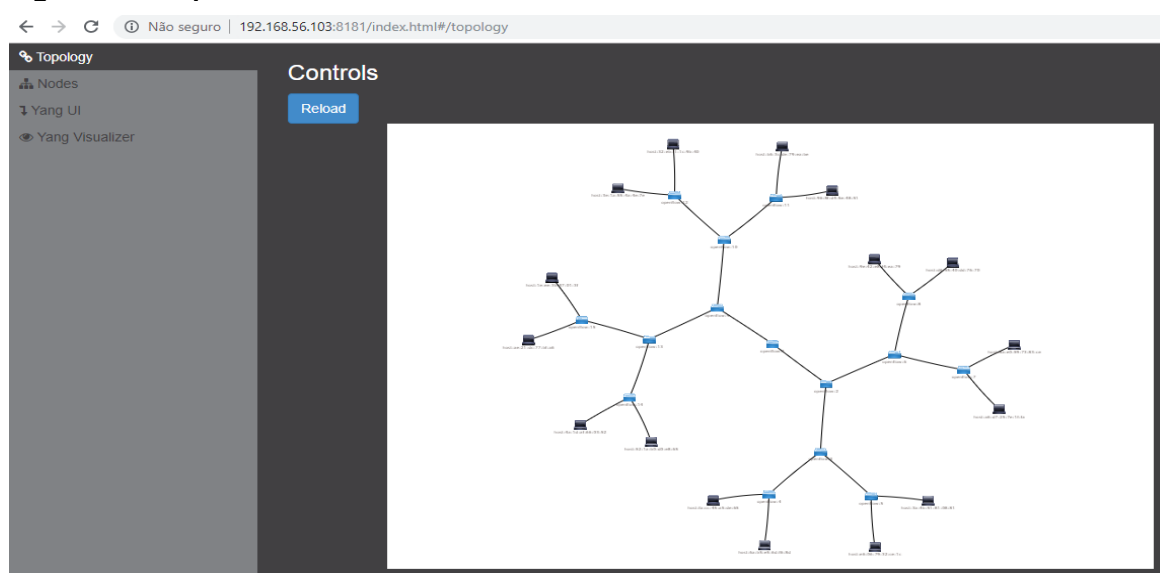
mininet@mininet-vm: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(s1, s2) (s1, s9) (s2, s3) (s2, s6) (s3, s4) (s3, s5) (s4, h1) (s4, h2) (s5
, h3) (s5, h4) (s6, s7) (s6, s8) (s7, h5) (s7, h6) (s8, h7) (s8, h8) (s9, s
10) (s9, s13) (s10, s11) (s10, s12) (s11, h9) (s11, h10) (s12, h11) (s12, h
12) (s13, s14) (s13, s15) (s14, h13) (s14, h14) (s15, h15) (s15, h16)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controller
c0
*** Starting 15 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 ...
*** Starting CLI:

```

Fonte: Autoria Própria

Após adicionar os dispositivos à rede, é possível visualizá-los via terminal, e também, via interface gráfica, conforme a Figura 19 abaixo:

Figura 19 - Dispositivos adicionados à rede Mininet



Fonte: Autoria Própria

Para testar a conexão e verificar se todos estão recebendo e enviando pacotes de dados entre eles, é possível executar o *ping* entre dois hosts diferentes, conforme mostra a Figura 20 e 21 abaixo:

Figura 20 - Ping entre hosts h1 e h9 na Topologia Ramificada

```

mininet@mininet-vm: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
mininet> h1 ping h9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=1.00 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.563 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.590 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=0.838 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=0.481 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=0.393 ms
64 bytes from 10.0.0.9: icmp_seq=7 ttl=64 time=0.583 ms
64 bytes from 10.0.0.9: icmp_seq=8 ttl=64 time=0.626 ms
64 bytes from 10.0.0.9: icmp_seq=9 ttl=64 time=0.532 ms
64 bytes from 10.0.0.9: icmp_seq=10 ttl=64 time=0.578 ms
64 bytes from 10.0.0.9: icmp_seq=11 ttl=64 time=0.579 ms
64 bytes from 10.0.0.9: icmp_seq=12 ttl=64 time=0.575 ms
64 bytes from 10.0.0.9: icmp_seq=13 ttl=64 time=0.897 ms
64 bytes from 10.0.0.9: icmp_seq=14 ttl=64 time=0.953 ms
64 bytes from 10.0.0.9: icmp_seq=15 ttl=64 time=0.989 ms
64 bytes from 10.0.0.9: icmp_seq=16 ttl=64 time=0.905 ms
64 bytes from 10.0.0.9: icmp_seq=17 ttl=64 time=0.634 ms
^C
--- 10.0.0.9 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16007ms
rtt min/avg/max/mdev = 0.393/0.689/1.009/0.192 ms
mininet>

```

Fonte: Autoria Própria

Figura 21 - Ping entre hosts h7 e h15 na Topologia Ramificada

```

mininet@mininet-vm: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
mininet> h7 ping h15
PING 10.0.0.15 (10.0.0.15) 56(84) bytes of data.
64 bytes from 10.0.0.15: icmp_seq=1 ttl=64 time=1.06 ms
64 bytes from 10.0.0.15: icmp_seq=2 ttl=64 time=0.346 ms
64 bytes from 10.0.0.15: icmp_seq=3 ttl=64 time=0.587 ms
64 bytes from 10.0.0.15: icmp_seq=4 ttl=64 time=0.564 ms
64 bytes from 10.0.0.15: icmp_seq=5 ttl=64 time=0.952 ms
64 bytes from 10.0.0.15: icmp_seq=6 ttl=64 time=0.681 ms
64 bytes from 10.0.0.15: icmp_seq=7 ttl=64 time=0.457 ms
64 bytes from 10.0.0.15: icmp_seq=8 ttl=64 time=0.586 ms
64 bytes from 10.0.0.15: icmp_seq=9 ttl=64 time=0.587 ms
64 bytes from 10.0.0.15: icmp_seq=10 ttl=64 time=0.831 ms
64 bytes from 10.0.0.15: icmp_seq=11 ttl=64 time=0.645 ms
64 bytes from 10.0.0.15: icmp_seq=12 ttl=64 time=0.604 ms
64 bytes from 10.0.0.15: icmp_seq=13 ttl=64 time=0.348 ms
64 bytes from 10.0.0.15: icmp_seq=14 ttl=64 time=0.600 ms
64 bytes from 10.0.0.15: icmp_seq=15 ttl=64 time=0.693 ms
64 bytes from 10.0.0.15: icmp_seq=16 ttl=64 time=0.572 ms
64 bytes from 10.0.0.15: icmp_seq=17 ttl=64 time=0.578 ms
^C
--- 10.0.0.15 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16007ms
rtt min/avg/max/mdev = 0.346/0.629/1.066/0.180 ms
mininet>

```

Fonte: Autoria Própria

Uma das funcionalidades no Mininet é poder criar vários tipos de topologias, por exemplo, a topologia linear, conforme a Figura 22 mostra:

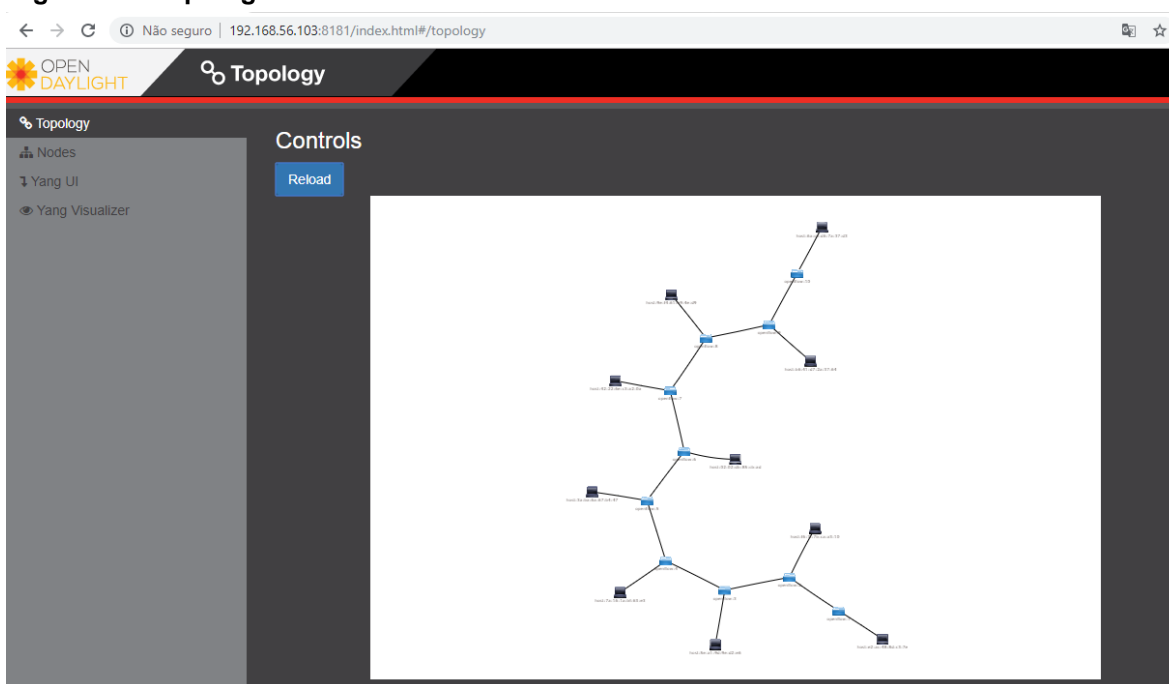
Figura 22 - Criação da Topologia Linear

```
mininet@mininet-vm: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (h6, s6) (h7, s7) (h8, s8) (h9, s9)
) (h10, s10) (s2, s1) (s3, s2) (s4, s3) (s5, s4) (s6, s5) (s7, s6) (s8, s7) (s9
, s8) (s10, s9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
*** Starting CLI:
```

Fonte: Autoria Própria

Pode-se confirmar a criação da topologia em ambos os modos, tanto via terminal quanto interface gráfica, conforme a Figura 23 mostra:

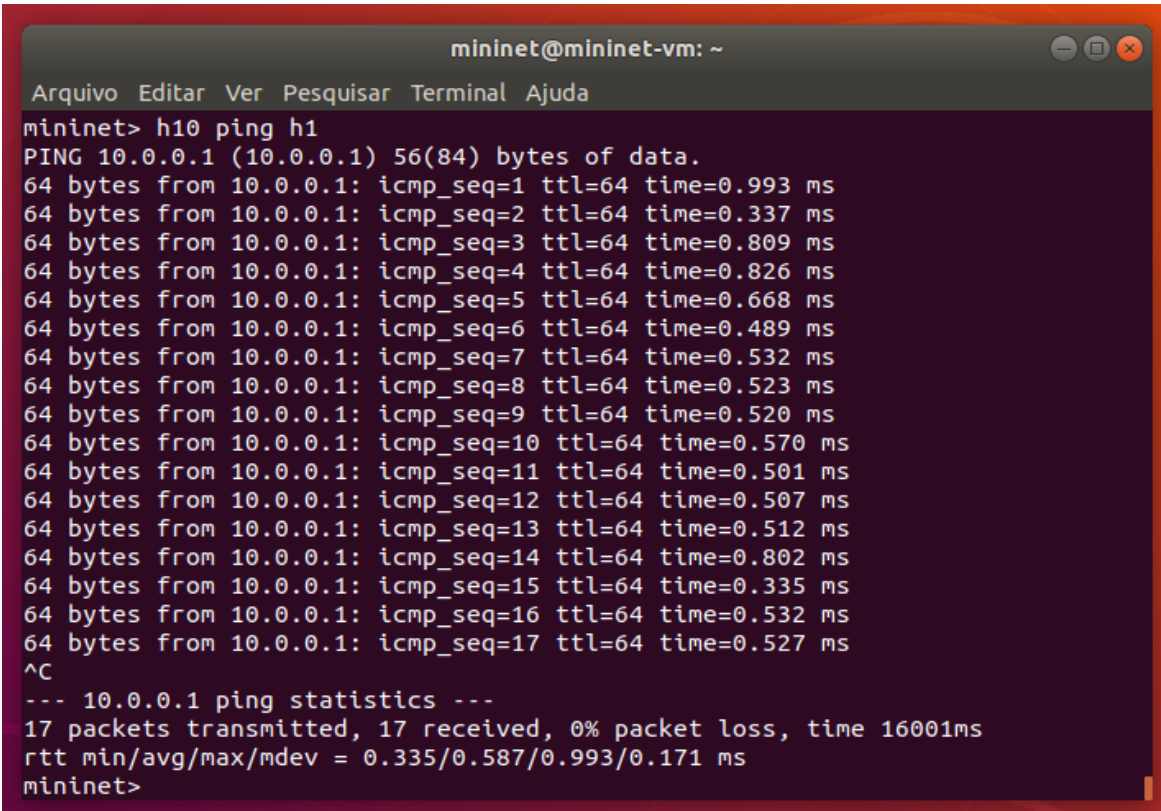
Figura 23 - Topologia Linear



Fonte: Autoria Própria

Para verificação, foi executado o comando *ping* novamente, conforme abaixo na Figura 24.

Figura 24 - Ping entre hosts h10 e h1



```
mininet@mininet-vm: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
mininet> h10 ping h1  
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.  
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.993 ms  
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.337 ms  
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.809 ms  
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.826 ms  
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.668 ms  
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.489 ms  
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.532 ms  
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.523 ms  
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.520 ms  
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.570 ms  
64 bytes from 10.0.0.1: icmp_seq=11 ttl=64 time=0.501 ms  
64 bytes from 10.0.0.1: icmp_seq=12 ttl=64 time=0.507 ms  
64 bytes from 10.0.0.1: icmp_seq=13 ttl=64 time=0.512 ms  
64 bytes from 10.0.0.1: icmp_seq=14 ttl=64 time=0.802 ms  
64 bytes from 10.0.0.1: icmp_seq=15 ttl=64 time=0.335 ms  
64 bytes from 10.0.0.1: icmp_seq=16 ttl=64 time=0.532 ms  
64 bytes from 10.0.0.1: icmp_seq=17 ttl=64 time=0.527 ms  
^C  
--- 10.0.0.1 ping statistics ---  
17 packets transmitted, 17 received, 0% packet loss, time 16001ms  
rtt min/avg/max/mdev = 0.335/0.587/0.993/0.171 ms  
mininet>
```

Fonte: Autoria Própria

É possível desativar e ativar um *switch* específico, caso seja necessário usando o comando “switch <nome do switch> stop”, e para ativar novamente, usa-se o mesmo comando, porém, adicionando “start” no final, a Figura 25 mostra a inativação do switch.

Figura 25 - Inativação do Switch



```
mininet> switch s10 stop  
mininet> █
```

Fonte: Autoria Própria

Por meio da interface gráfica, pode-se obter algumas informações referente aos *switches*, como o fluxo, nós, número de portas ativas em cada *switch*. Neste local também são listados todos os *switches* ativos na rede, conforme a Figura 26 mostra:

Figura 26 - Lista de Switches Ativos

Node Id	Node Name	Node Connectors	Statistics
openflow:1	None	3	Flows Node Connectors
openflow:6	None	4	Flows Node Connectors
openflow:7	None	4	Flows Node Connectors
openflow:8	None	4	Flows Node Connectors
openflow:9	None	4	Flows Node Connectors
openflow:2	None	4	Flows Node Connectors
openflow:3	None	4	Flows Node Connectors
openflow:4	None	4	Flows Node Connectors
openflow:5	None	4	Flows Node Connectors

Fonte: Autoria Própria

No *hyperlink Node Connectors* é possível visualizar informações como a quantidade de pacotes recebidos e enviados, assim como a quantidade em *bytes* de cada porta de um determinado *switch*. A Figura 27 exemplifica.

Figura 27 - Node Connectors

Node Connector Statistics for Node Id - openflow:1													
Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions	
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0	
openflow:1:2	1340	1052	109006	88846	0	0	0	0	0	0	0	0	
openflow:1:1	42	1340	2996	109006	0	0	0	0	0	0	0	0	

Fonte: Autoria Própria

5 CONCLUSÃO

Com a elaboração deste trabalho, pode-se concluir que o gerenciamento da rede fica mais simples, mais flexível, programável, com um maior poder de integração, pois está centralizado em um controlador lógico, possibilitando a criação de uma topologia adequada para uso comercial/empresarial.

Com a virtualização é possível romper os obstáculos que podem vir a surgir, prever acontecimentos e tomar decisões antecipadas, obtendo os testes necessários para levar o projeto de homologação para produção.

Ainda com a virtualização é possível colocar em prática muitas ideias, topologias, teorias, para alcançar o objetivo principal de cada usuário. É de grande valor mencionar que os *softwares* usados são todos *open source*, cada um com sua grande comunidade para dar suporte e tirar dúvidas. Caso seja identificado alguma melhoria é possível encaminhar a mesma para cada comunidade em específico.

5.1 TRABALHOS FUTUROS

Com o grande avanço da tecnologia, é possível usufruir muito mais das aplicações do controlador ODL, como por exemplo, uma forma de fazer os roteamentos, os balanceamentos de carga, aplicações de políticas de segurança, clusterização, entre outros.

REFERÊNCIAS

CHAYAPATHI, R.; HASSAN, S. F.; SHAH, P. Network functions virtualization (NFV) with a touch of sdn. **Pearsoncmg**, 2016. Disponível em: <bit.ly/2NhAUeJ>. Acesso em: 15 ago 2019.

CISCO. Meraki Documentation. **Documentation Meraki**, 2018. Disponível em: <bit.ly/2q0Dz4k>. Acesso em: 30 ago 2019.

CISCOPRESS. CISCOPRESS. **Metro Internet Services**, 2003. Disponível em: <<http://bit.ly/32XOo5W>>. Acesso em: ago 30 2019.

COSTA, L. R. OpenFlow e o paradigma de redes definidas por *software*. **Universidade de Brasília Biblioteca Central**, 16 jul 2013. Disponível em: <<http://bit.ly/2MTq1AL>>. Acesso em: 1 set 2019.

EFREMOVA, L.; ANDRUSHKO, D. What's in OpenDaylight. **Mirantis**, 2015. Disponível em: <<http://bit.ly/31V3yHQ>>. Acesso em: 1 set 2019.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN: An Intellectual History. **princeton**, 2013. Disponível em: <<http://bit.ly/31RUGml>>. Acesso em: ago 15 2019.

FLOODLIGHT. Floodlight is an open SDN controller. **Project Floodlight**, 2019. Disponível em: <<http://bit.ly/2WmcEwf>>. Acesso em: 1 set 2019.

GÖRANSSON, P.; BLACK, C. **Software Defined Networks: A Comprehensive Approach**. 1. ed. Waltham: Morgan Kaufmann Publishers, 2014.

GUEDES, D. et al. Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento das pesquisas em Redes de Computadores. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, Ouro Preto, 30(4) 2012. 1-47.

KIRKPATRICK, K. Software-Defined Networking. **Communications of the ACM**, New York, v. 56, n. 9, p. 16-19, set 2013. DOI:10.1145/2500468.2500473.

MCKEOWN, N. et al. Openflow: Enabling Innovation in Campus Networks. **ACM SIGCOMM Computer Communication Review**, Seattle?, v. 38, n. 2, p. 69-74, abr 2008. DOI: 10.1145/1355734.1355746.

MEDVED, J. et al. OpenDaylight: Towards a Model-Driven SDN Controller Architecture. **Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks**, Sydney, 19 jun 2014. 1-6.

MININET. Mininet. **An Instant Virtual Network on your Laptop (or other PC)**, 2018. Disponível em: <<http://bit.ly/34d4JUUm>>. Acesso em: 1 set 2019.

ONF. Openflow-enabled SDN and Network Functions Virtualization. **ONF**, 2014. Disponível em: <<http://bit.ly/2Jvtl2Q>>. Acesso em: 1 set 2019.

ONOS. ONOS is building a better network. **Onosproject**, 2019. Disponível em: <<http://bit.ly/31Wa9Sc>>. Acesso em: 1 set 2019.

ROTHENBERG, C. E. et.al Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia**, Campinas, v. 7, n. 1, p. 65-76, jul 2010. Acesso em: 1 set 2019. p. 65-76.

S. BADOTRA; J. SINGH. Opendaylight as a controller for software defined networking. **International Journal of Advanced Research in Computer Science**, Punjab, 2017. Disponível em: <bit.ly/2Jo3CcL>. Acesso em: 15 ago. 2019. ISSN 0976-5697.

SDX. What is Ryu Controller?. **Sdxcentral**, 2016. Disponível em: <<http://bit.ly/2Njl41X>>. Acesso em: 1 set 2019.

SEZER, S. et al. Are we ready for SDN? Implementation Challenges for Software-Defined Networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36-43, jul 2013. ISSN 0163-6804.

SILVA, J. P. S. A. E.; FARIA, P. R. Redes Definidas por Software: Uma abordagem inicial ao conceito e prática. **Biblioteca Digital Anton Darkitsch**, 2017. Disponível em: <<http://bit.ly/2JxVW7U>>. Acesso em: 1 set 2019.

VIRTUALBOX. Oracle. **Virtualbox**, 2019. Disponível em: <<http://bit.ly/32VpaFd>>. Acesso em: 1 set 2019.