

# Redes definidas por software para o controle de fluxo de dados em redes locais virtuais

João Pedro de França Lourenço<sup>1</sup>, Ricardo Cesar Câmara Ferrari<sup>2</sup>

<sup>1</sup>Discente do curso superior Bacharelado em Ciência da Computação – Instituto Federal de Educação Ciência e Tecnologia de São Paulo, Campus Presidente Epitácio

<sup>2</sup>Docente do curso superior Bacharelado em Ciência da Computação – Instituto Federal de Educação Ciência e Tecnologia de São Paulo, Campus Presidente Epitácio  
Rua José Ramos Júnior, 27-50 - 19470-000 Presidente Epitácio, SP - Brasil

joao.franca@aluno.ifsp.edu.br<sup>1</sup>, ricardo.ferrari@ifsp.edu.br<sup>2</sup>

**Resumo.** *Este trabalho tem como objetivo realizar um estudo sobre redes definidas por software (SDN) com o intuito de desenvolver uma solução computacional para o gerenciamento de redes locais virtuais. Dessa forma, analisar o comportamento do fluxo de dados para estabelecer seu controle adequado por meio da criação de redes virtuais locais. Com os experimentos desenvolvidos, foi possível realizar o controle de uma rede definida por software, como a captura de dados pelo controlador POX, gerenciamento dos switches virtuais e a criação de uma topologia de rede utilizando o Mininet e Open vSwitch, além de analisar o funcionamento do protocolo OpenFlow.*

**Abstract.** *The aim of this work is to study software-defined networks (SDN) in order to develop a computer solution for managing virtual local networks. In this way, the behavior of the data flow can be analyzed in order to establish proper control through the creation of virtual local networks. With the experiments developed, it was possible to control a software-defined network, such as data capture by the POX controller, management of virtual switches and creation of a network topology using Mininet and Open vSwitch, as well as analyzing the operation of the OpenFlow protocol.*

## 1. Introdução

Nos dias atuais, o mundo está cada vez mais conectado devido aos avanços contínuos da tecnologia da informação e comunicação. É difícil imaginar a vida cotidiana sem a presença das redes de computadores, que estão presentes em todos os aspectos da vida humana, desde o lazer até os negócios. Um exemplo disso pode ser visto no Brasil, onde um estudo conduzido pelo Instituto Brasileiro de Geografia e Estatística (IBGE, 2021), revelou que aproximadamente 90,0% dos lares do país têm acesso à internet, representando um aumento significativo de 6 pontos percentuais em relação a 2019, quando 84,0% dos lares estavam conectados à grande rede.

Além disso, com o surgimento de novas tecnologias como vídeos em alta resolução (4k e 8k), dispositivos inteligentes para uso doméstico e realidade aumentada, há uma crescente demanda por redes de computadores mais programáveis e com melhor desempenho. Nesse cenário, a comunidade acadêmica está buscando soluções que possam atender a essas necessidades, como as Redes Definidas por *Software* ou Software Defined Network (SDN), que fazem uso do protocolo OpenFlow para consultar uma tabela de fluxo e assim separar o plano de dados do plano de controle.

Este trabalho tem como objetivo fazer o gerenciamento de redes locais virtuais (VLANs) através de uma solução computacional. Para isso, é realizada uma

implementação no controlador POX. Para contextualizar, o POX é um controlador de redes definida por software onde permite gerenciar e manipular o fluxo de dados em redes de forma programática. Sendo assim, permitindo o gerenciamento das VLANs para o controle de fluxos de dados e a criação de um protótipo web para que possa realizar o gerenciamento das VLANs. Ao separar o plano de controle do plano de dados, o controlador POX possibilita uma gestão das redes locais virtuais, permitindo ajustes conforme as necessidades da rede.

## **2. Fundamentação Teórica**

Nesta seção é apresentada a fundamentação teórica sobre os principais conceitos abordados para o desenvolvimento do projeto. Iniciando com uma visão geral de redes de computadores, redes locais virtuais, redes definidas por *software* (SDN), o protocolo openflow, switch virtual e por fim sobre o controlador SDN.

### **2.1 Visão Geral de Redes de Computadores**

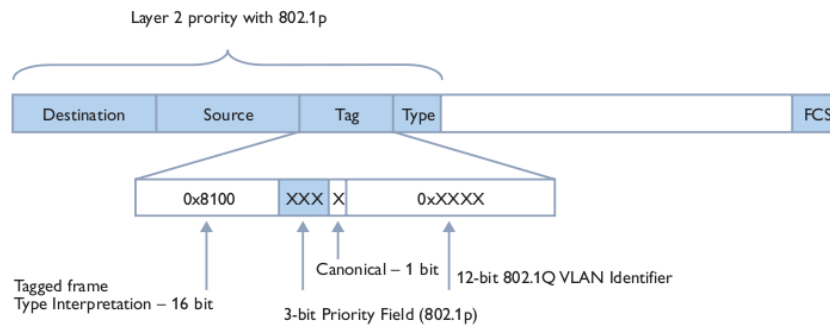
Redes de computadores podem ser divididas em três planos de funcionalidades: Plano de Dados, referente aos dispositivos físicos da rede e ao repasse de dados; Plano de Controle, que diz respeito aos protocolos utilizados para o povoamento das tabelas de fluxo; Plano de Gerenciamento, responsável pelo monitoramento remoto e pelas configurações de funcionalidades, incluindo serviços de software, como, por exemplo, as ferramentas SNMP (*Simple Network Management Protocol* - Protocolo Simples de Gerência de Rede). O plano de Gerenciamento define as políticas da rede, o Plano de Controle reforça essas políticas e o Plano de Dados encaminha os pacotes pela rede.

### **2.2 Redes locais virtuais**

Em 1996, foi aprovado o padrão 802.1Q, também conhecido como VLAN (*Virtual Local Area Network*) *Tagging*, que possibilita o uso de redes locais virtuais em equipamentos de diferentes fabricantes. Este padrão abriu caminho para padronizações adicionais relacionadas às VLANs. Uma VLAN pode ser entendida como uma rede local configurada por *software* sobre a infraestrutura física da rede. A vantagem das VLANs é permitir a segmentação lógica de uma LAN, criando segmentos individuais com seu próprio domínio de colisão e *broadcast*. Ao mover um dispositivo de uma VLAN para outra, apenas a configuração lógica é alterada por *software*, mantendo-se a conexão física intacta. No contexto das redes Ethernet, o padrão 802.1Q capacita os *switches Ethernet* a segmentar a rede logicamente em VLANs. Isso significa que as VLANs são configuradas nos *switches Ethernet*, sem necessidade de configurações adicionais nos dispositivos terminais, como computadores, servidores, câmeras e telefones IP, que se comunicam na rede.

Os quadros que trafegam entre os *switches* podem conter um cabeçalho adicional conhecido como etiqueta (*tag*), localizado nos quatro *bytes* seguintes ao campo de endereço de origem. (TANENBAUM; ANDREW S, 2021).

**Figura 1. Tag inserida em um quadro Ethernet.**



Fonte: SEGMENTAÇÃO de LANs e VLANs (2020).

Nos padrões IEEE 802.1Q, os dois primeiros *bytes* da etiqueta VLAN são conhecidos como Rótulo de Identificação de Protocolo (*Tag Protocol Identifier* - TPID), com um valor fixo de 0x8100h. Os dois *bytes* seguintes compõem a Informação de Controle da Etiqueta (*Tag Control Information* - TCI). Esses componentes são essenciais para distinguir e diferenciar as VLANs em redes que implementam essa tecnologia, conforme descrito por Kurose e Ross (2021). Dentro da TCI está contido o Identificador de VLAN (*Vlan Identifier* - VID), um número que indica a qual VLAN cada quadro *Ethernet* pertence. Esse identificador é crucial para o correto encaminhamento dos quadros dentro da rede VLAN.

As portas do switch que recebem e transmitem quadros com o cabeçalho de etiqueta VLAN operam em modo *tagged*. Em contraste, as portas configuradas para operar em modo *untagged* não modificam ou adicionam cabeçalhos VLAN aos quadros. A adição desse cabeçalho é especialmente necessária quando os quadros trafegam entre switches *Ethernet*, que são essenciais na definição da topologia VLAN. (TANENBAUM; ANDREW S, 2021).

Conforme ilustrado na Figura 2, o cabeçalho TCI reserva 12 *bits* para o número identificador de VLAN (VID), permitindo a configuração de até 4096 VLANs simultaneamente em uma mesma rede local física. Em cenários de grande escala, como data *centers* extensos, a capacidade de 4096 VLANs pode ser insuficiente.

**Figura 2. Tag Control Information (TCI).**



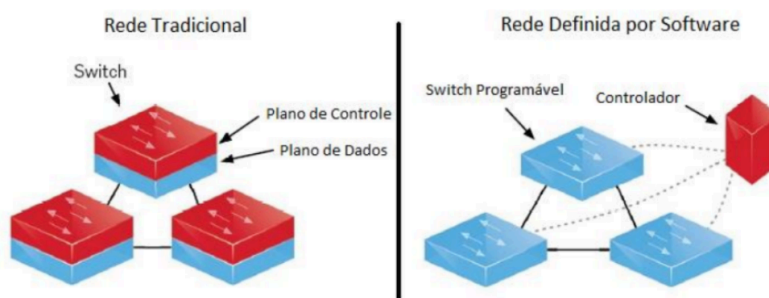
Fonte: IEEE COMPUTER SOCIETY, 2005.

## 2.3 Redes Definidas por Software - SDN

Com as redes tradicionais, a evolução da rede de computadores trouxe desafios complexos de gerenciamento e dificuldades para atender às demandas atuais. Diante disso, surgiu a abordagem inovadora das Redes Definidas por *Software* (SDN). Essa abordagem envolve a programação dos *switches* utilizados nas redes de dados de uma maneira completamente nova e fundamental (GÖRANSSON; BLACK; CULVER,

2016). A SDN é considerada uma solução eficiente, pois permite centralizar o controle da rede e retirar essa responsabilidade dos dispositivos de *hardware*, como *switches* e roteadores. Sua arquitetura propõe uma nova forma de programar e controlar os *switches* usados em redes de dados modernas, com a separação dos planos de dados e controle. Nesse modelo, um controlador SDN centralizado logicamente assume a responsabilidade de gerenciar o tráfego da rede, impor regras e especificações, conforme (Mafioletti, 2015). Em contraste, nas redes tradicionais, o plano de controle e o plano de dados estão integrados, ou seja, o plano de controle é executado no próprio *hardware*, não podendo realizar tomadas de decisões diferentes das já pré-estabelecidas pelos protocolos instalados no seu sistema operacional. O plano de controle é responsável por toda a informação da rede, direcionando os pacotes, criando rotas de fluxo e gerenciando algoritmos de roteamento. O plano de dados, por sua vez, é encarregado apenas do encaminhamento dos pacotes de acordo com as regras ou especificações estabelecidas pelo plano de controle (NADEAU e GRAY, 2013). Dito isso, a Figura 3 apresenta a arquitetura de uma rede SDN e de uma rede tradicional.

**Figura 3. Diferença entre rede tradicional e SDN.**



Fonte: PRAJAPATI; SAKADASARIYA; PATEL, 2018.

Na Figura 3, é possível observar que na arquitetura da Rede Definida por Software (SDN), localizada no lado direito, o plano de controle, representado pela cor vermelha, está separado do plano de dados, que consiste nos *switches* em azul. Além disso, vale destacar que essa separação permite que um único controlador gerencie vários *switches* em uma rede, proporcionando facilidade na configuração e controle, já que as políticas e configurações podem ser aplicadas de forma centralizada e repassadas para os *switches*.

## 2.4 OpenFlow

Com o entendimento sobre redes definidas por *software* (SDN), assim como em qualquer rede de computadores, essas redes também dependem de protocolos para estabelecer a comunicação. Nesse caso, é essencial que haja suporte ao protocolo *OpenFlow*, desenvolvido em 2008 por um grupo de pesquisadores da Universidade de Stanford, liderado por McKeown et al. (2008), como uma proposta de protocolo flexível que pudesse trabalhar em conjunto com a tecnologia SDN, possibilitando a conexão entre o controlador SDN e o *switch* virtual ou roteador. Conforme mencionado por Göransson, Black e Culver (2016), esse protocolo consiste em um conjunto de mensagens trocadas entre os componentes da rede SDN. Através dessas mensagens, o controlador é capaz de programar o *switch* para executar diferentes ações, como definir, modificar ou excluir fluxos. Um fluxo pode ser compreendido como um conjunto de

pacotes que são transferidos de um ponto de extremidade da rede para outro. Esses pontos de extremidade podem ser definidos por endereços IP, portas TCP/UDP, pontos de extremidade VLAN, túneis de camada três e portas de entrada, entre outros. Um conjunto de regras é responsável por orientar o *switch* no encaminhamento apropriado daquele fluxo, de acordo com as especificações definidas pelo controlador. Segundo Silva (2017), os tipos de mensagens mais comuns do protocolo OpenFlow podem ser visualizados na Tabela 1.

**Tabela 1 - Mensagens OpenFlow.**

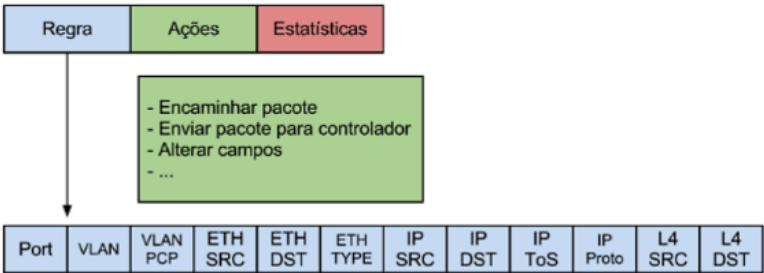
<b>Mensagem</b>	<b>Origem</b>	<b>Destino</b>	<b>Descrição</b>
Flow-mod	Controlador	<i>Switch</i>	Utilizada pelo controlador para o gerenciamento das tabelas de fluxo.
PacketIn	<i>Switch</i>	Controlador	<i>Switch</i> requisita ao controlador o que deve ser feito com o pacote.
Flow-out	Controlador	<i>Switch</i>	Pacotes enviados pelo controlador através do <i>switch</i> para rede. É especificado a porta do <i>switch</i> para o envio.
Flow-stats	Ambos	Ambos	Usadas na coleta de estatísticas do <i>switch</i> .

Fonte: Adaptada de SILVA (2017).

Sabendo os tipos de mensagens mais comuns do protocolo, é importante entender como são armazenadas as regras enviadas pelo controlador para o *switch*. Dito isso, entra-se na chamada tabela de fluxo. As tabelas de fluxo possuem várias entradas que especificam regras de correspondência, ações a serem realizadas em pacotes que correspondem a essas regras, onde são guardadas as *flow entries*, ou entradas de fluxo. *Flow entries* são estruturas de dados que definem como os pacotes devem ser processados enquanto atravessam um *switch* de rede. Tais entradas de fluxo armazenam três campos, segundo Ono et al., (2020): as regras de correspondência, a ação a ser tomada e os contadores. Essas regras, que também podem ser chamadas de regras de correspondência, são definidas com base nos atributos do cabeçalho do pacote. Quando um pacote chega a um *switch*, ocorre uma verificação para determinar se algum campo do cabeçalho corresponde a uma dessas regras. (KREUTZ et al., 2015). A Figura 4

mostra os campos do cabeçalho que podem ser usados para a comparação com as regras.

**Figura 4. Campos do cabeçalho do pacote na comparação com as regras.**



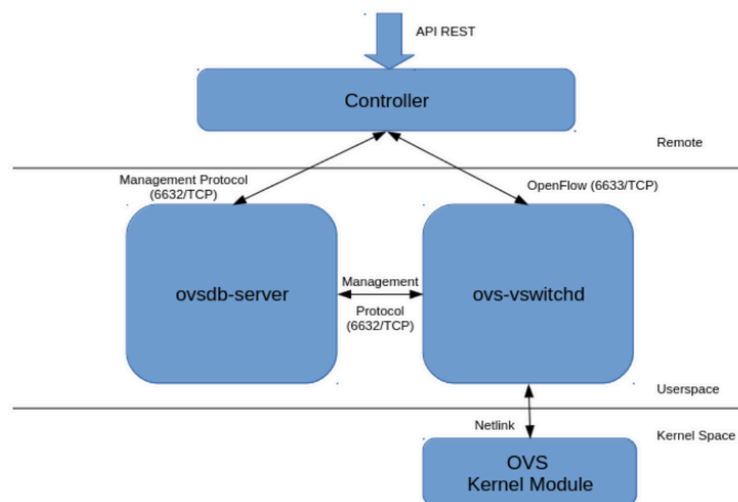
Fonte: RODRIGUES COSTA, 2013.

A ação, por sua vez, é o que será feito com aquele pacote que teve sua comparação bem sucedida. Para ilustrar, considere que desejasse que todos os pacotes com o campo IP de origem igual a 192.168.1.115 sejam direcionados para a porta 5 do *switch*. Para isso, é necessário criar uma entrada de fluxo na tabela do *switch*, por meio do controlador, com a regra de correspondência do IP de origem igual a 192.168.1.115. Dessa forma, qualquer pacote que possua esse valor no campo de IP de origem será encaminhado de acordo com a ação definida na entrada de fluxo, que neste caso seria o envio para a porta 5 do *switch*. Por fim, vale ressaltar que os campos descritos das entradas de fluxo e os atributos do cabeçalho do pacote compatíveis com as regras descritos na Figura 4 são da versão 1.0 do OpenFlow e podem mudar de versão para versão, neste trabalho foi apresentado esse exemplo, pois é o suportado pelo controlador POX, que foi o controlador escolhido.

### 2.5 Switch Virtual

Após o entendimento sobre o protocolo OpenFlow, para que o controlador SDN possa se comunicar com um *switch* que não é compatível com o OpenFlow de fábrica, como não possuímos um *switch* físico é necessário criar um *switch* virtual que seja compatível. Atualmente, a opção mais utilizada é o *Open vSwitch*, um *switch* virtual multicamada projetado para permitir a automação de rede em larga escala por meio de extensão programática. Além disso, ele oferece suporte a interfaces e protocolos de gerenciamento padrão, como *OpenFlow*, *NetFlow* e *sFlow*, entre outros *kernel* (OPENVSWITCH, 2024). Na Figura 5, a arquitetura do *Open vSwitch* é apresentada, destacando seus três principais componentes, o *kernel module*, *ovsdb-server* e o *ovs-vswitchd*.

**Figura 5 - Arquitetura do Open vSwitch.**



Fonte: Fernandez e Muñoz (2015).

Segundo Mafioletti (2015), o uso do módulo do *kernel* no *Open vSwitch* permite que ele funcione como um componente integrado ao *kernel* Linux. Essa integração é importante, pois possibilita uma colaboração eficiente com outros componentes do sistema operacional, resultando em benefícios significativos em termos de desempenho. O *ovs-vswitchd* é responsável por gerenciar todos os *switches* virtuais na máquina local, bem como se comunicar com o controlador, *ovssdb-server* e o módulo do *kernel* através dos protocolos *OpenFlow*, *OVSDb* e *Netlink*, respectivamente, como se pode ver na Figura 5. O *ovssdb-server*, por sua vez, provê interfaces *RPC* (*Remote Procedure Call*) para um ou mais bancos de dados do *Open vSwitch*. Esses bancos de dados armazenam informações de configuração do *switch*, como *bridges*, interfaces e endereços do controlador *OpenFlow*, entre outras configurações. Além disso, este servidor interage com os gerenciadores do *OVSDb* e o *ovs-vswitchd* utilizando o protocolo *OVSDb* (FERNANDEZ E MUÑOZ, 2015).

## 2.5 Controlador SDN

O controlador é o elemento central das decisões de processamento de fluxos em uma rede que implementa SDN/OpenFlow, pois é através dele que o administrador da rede define como se comportam certos encaminhamentos ou descartes de dados em cenários específicos. As tabelas de fluxo, que determinam as decisões de um *switch*, são modeladas de acordo com as regras programadas no controlador e atualizadas sempre que necessário. Segundo Passos et al. (2016), o NOX foi o primeiro controlador SDN, o NOX possui duas versões distintas: o NOX clássico, que suporta as linguagens C++ e Python, e o "Novo NOX", exclusivamente compatível com C++, destacando-se por ter um código base otimizado e um desempenho superior em comparação ao seu antecessor. Com o intuito de facilitar e tornar o desenvolvimento mais flexível, foi criado o POX, uma versão "irmã" do NOX completamente escrita em Python, destacando-se como a opção mais popular para controladores SDN. Tendo isso em vista, o controlador POX foi escolhido para o desenvolvimento deste trabalho.

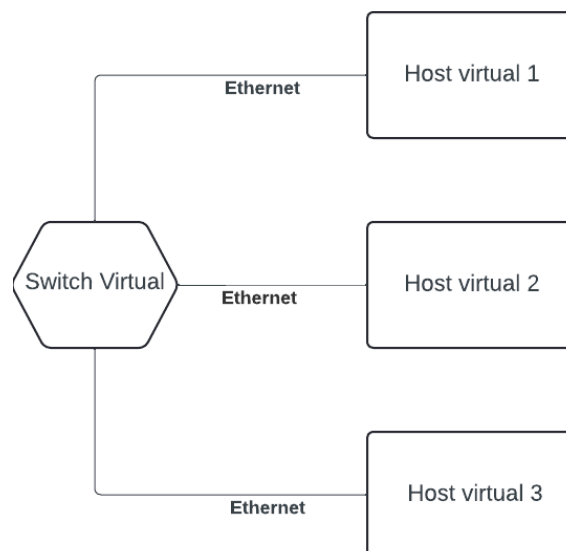
### 3. Desenvolvimento

Nesta seção, é apresentado o desenvolvimento, experimentos e testes necessários para garantir a configuração e o funcionamento de uma rede definida por software (SDN) utilizando o controlador POX. Inicialmente, foi feita a preparação da topologia de rede, seguida pela captura de informações pelo controlador. Em seguida, foi desenvolvida a estrutura do banco de dados e estabelecida a conexão com o POX, permitindo a interação entre o controlador e a base de dados. Além disso, foi realizado um experimento para a criação de VLANs através do POX e, por fim, foi desenvolvido um protótipo web.

#### 3.1 Preparação da topologia de rede

É montada a arquitetura da rede com um *switch* virtual e três *hosts* virtuais. Na Figura 6, o mininet está sendo utilizado para a criação de três *hosts* virtuais e um switch virtual Open vSwitch.

**Figura 6 - Arquitetura da rede.**



Fonte: Elaborado pelo próprio autor.

A especificação do equipamento utilizado nos experimentos e testes está especificado na Tabela 2.

**Tabela 2 - Especificações da máquina virtual.**

Processador	Intel core i5-4460 @ 3.20 GHz x 4
Sistema Operacional	Ubuntu 20.04 LTS
Tipo do Sistema Operacional	64 bits
Memória	4 GB



Foi realizada uma conexão via SSH a máquina virtual especificada na Tabela 2. Após feita a conexão foi realizada a configuração e a criação da topologia utilizando a ferramenta de emulação de rede Mininet com o seguinte comando:

- `mn --topo single,3 --mac --switch ovsk --controller remote`
- Parâmetro 1: `--topo single,3`
- Parâmetro 2: `--mac`
- Parâmetro 3: `--switch ovsk`
- Parâmetro 4: `--controller remote`

Com o comando `mn` é iniciado o Mininet e com alguns parâmetros necessários para a configuração da topologia. Os parâmetros utilizados para este teste foram os seguintes. No parâmetro 1 define a topologia da rede como uma topologia simples com três *hosts* e um switch. Isso significa que há um *switch* virtual conectado a três *hosts* diretamente. O parâmetro 2 define que os endereços MAC serão usados para identificar os dispositivos de rede. Isso é importante para a comunicação entre dispositivos na rede. No parâmetro 3, Especifica que os *switches* serão do tipo Open vSwitch. No parâmetro 4, Especifica que o controlador será configurado externamente, ou seja, não será utilizado o controlador padrão do mininet. Assim o Mininet será configurado automaticamente para se conectar a um controlador externo na porta padrão 6653. Na Figura 7 foi inicializado o Mininet, com isso é apresentada toda a criação da topologia, como a criação dos *hosts* virtuais (h1, h2, h3), a criação do switch virtual (s1) e a criação dos links, onde cada *host* é conectado a uma porta do *switch* virtual. Logo em seguida é inicializada a conexão com o controlador, mas não obteve sucesso pois o controlador ainda não está sendo executado.

**Figura 7 - Criação da topologia utilizando o Mininet.**

```
vagrant@control-node:~$ sudo -E mn --topo single,3 --mac --switch
ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
mininet> 
```

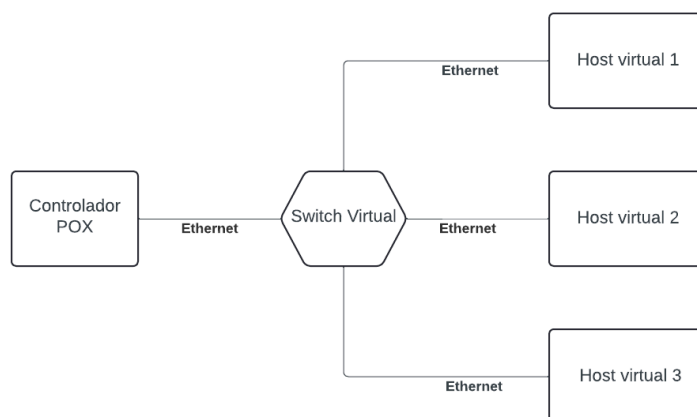
Fonte: Elaborado pelo próprio autor.

### 3.2 Captura de informações com o controlador POX

Nesta seção é preparado e configurado o controlador POX para que possa dar continuidade nos testes e a captura de informações com o controlador POX. O objetivo

principal desse teste é coletar informações da rede por meio do uso do componente 'flow\_stats' do controlador POX.

**Figura 8 - Topologia com o controlador POX.**



Fonte: Elaborado pelo próprio autor.

O controlador POX é executado na mesma máquina virtual onde estão os *hosts* virtuais e switch virtual. Em seguida é instalado o controlador POX junto com seus componentes, incluindo o 'flow\_stats', que trabalha captando estatísticas do fluxo que passa pelo *switch*, porém para esse teste foi criado um novo componente chamado 'experimento-1' a partir do componente 'flow\_stats', para capturar o protocolo se ele for TCP e UDP, id da VLAN, endereço MAC de origem e destino, IP de origem, IP de destino, e a porta do *switch* virtual. O comando './pox.py forwarding.l3\_learning experimento-1' foi executado no computador, o componente 'forwarding.l3\_learning' faz com que ocorra o encaminhamento de pacotes em uma rede baseada em camada 3, junto com o módulo personalizado para apresentar os dados dos pacotes do fluxo da rede. Foi executado os comandos iperf e iperfudp para gerar fluxo na rede e assim capturar os pacotes com o POX.

### 3.3 Criação e conexão ao Banco de Dados

Nesta seção, é realizada a criação e conexão ao banco de dados para o armazenamento de informações do controlador POX. Como id da VLAN e porta do *switch* virtual. O objetivo é realizar uma integração do controlador POX com um banco de dados, para que tudo esteja pronto para implementações futuras, como por exemplo o sistema *web*. É utilizado o banco de dados relacional PostgreSQL. A base de dados é criada, juntamente com uma Tabela para armazenar as informações necessárias para o controlador POX. A Tabela 'vlan\_ports' possui quatro colunas, Id, Id da vlan, porta do *switch* e a data de quando foi criada a vlan. Em seguida é inserida na tabela as informações e gravadas na base de dados. Com essas informações gravadas já é possível consultá-las utilizando o controlador POX.

Em seguida, é criado um método no componente 'experimento\_db.py' no controlador POX, esse método é responsável por se conectar na base de dados e obter as informações necessárias, como, identificador da VLAN e porta do *switch* virtual. Com essas informações já é possível que o controlador faça a gravação na tabela de fluxo do

switch, com as VLANs determinadas nas portas do switch. O controlador é executado utilizando novos parâmetros para se conectar a base de dados.

```
./pox.py forwarding.l3_learning vlan_controller_db --host=192.168.0.124  
--dbname=sdn --user=docker --password=docker
```

Os parâmetros contendo o IP da base de dados e algumas informações adicionais como, nome da base de dados, usuário e senha são necessários para realizar o experimento.

### 3.4 Criação das VLANs com o POX

É realizada a criação das VLANs nos switches virtuais utilizando o POX. Com a base de dados sendo executada, e o controlador POX acessando esses dados, é possível realizar a configuração das VLANs. Após consultar o banco de dados, é criado um método no componente ‘experimento\_db.py’, esse método faz uma consulta ao banco de dados para capturar o id da VLAN e a porta do *switch*. Em seguida é realizada a gravação na tabela de fluxo do *switch* virtual, informando o id da VLAN e em qual porta do *switch* ela pertence. Foi criada duas VLANs, 10 e 20 em quatro portas no switch, cada porta é atribuída a uma VLAN.

### 3.5 Criação do protótipo Web

O objetivo é criar um protótipo web para gerenciar as VLANs utilizando o microframework Flask. É criada uma página web para gerenciar os dados de VLANs. Foi utilizado o Flask para realizar a criação do protótipo, realizando a configuração inicial para que ele possa se comunicar com a base de dados. Essa conexão é feita a partir das bibliotecas ‘Flask-SQLAlchemy’ com a extensão do ‘psycopg2’, para realizar as consultas no banco de dados PostgreSQL. Com a configuração da conexão do banco de dados realizada, foi criada uma página principal utilizando HTML, com um formulário para preencher os dados das VLANs e uma tabela para apresentar as portas do switch com suas respectivas VLANs. Os métodos para realizar as operações CRUD estão em um script Python, onde é utilizado o Flask para criar um servidor WEB.

## 4. Resultados

Nesta seção serão apresentados os resultados obtidos dos experimentos descritos na seção anterior.

### 4.1. Resultados da Captura de informações com o controlador POX

O componente modificado do controlador POX permite a captura dos pacotes que passam pelo switch virtual e a visualização do protocolo, se ele é TCP ou UDP, IP de origem, IP de destino, endereço MAC de origem e destino, id da VLAN e a porta do switch virtual. A Figura 9 mostra a utilização do comando iperf no mininet, onde é enviado de um host ao outro um fluxo TCP e UDP sendo assim possível capturar esses fluxos com o controlador POX.

**Figura 9 - Envio de pacotes TCP e UDP entre os hosts.**

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['21.2 Gbits/sec', '21.2 Gbits/sec']
mininet> iperfudp 5m h1 h3
*** Iperf: testing UDP bandwidth between h1 and h3
*** Results: ['5m', '5.00 Mbits/sec', '5.00 Mbits/sec']
mininet> □
```

7

Fonte: Elaborado pelo próprio autor.

Os fluxos necessários para o POX capturar as informações, foi utilizado o comando `iperf` que está disponível nativamente no `mininet`. Com ele é possível fazer o envio de pacotes TCP ou UDP de um host a outro. Os comandos para a realização do envio dos pacotes foram utilizados da seguinte maneira.

- Comando 1: `iperf h1 h3`
- Comando 2: `iperfudp 5m h1 h3`

Pode-se observar no comando 1, para ser realizado o envio de pacotes TCP, é necessário apenas passar por parâmetro o host de origem e de destino. Com isso será enviado pacotes TCP entre o host 1 e o host 3. O comando 2 é bem parecido com o comando 1, a diferença é que será enviado pacotes UDP e é preciso informar por parâmetro a largura de banda que queremos utilizar, no experimento foi utilizado apenas 5Mb por segundo. Gerando um fluxo com esses pacotes já é possível capturar com o controlador POX as informações necessárias, como exemplificado na Figura 10.

**Figura 10 - Saída adquirida utilizando o POX com o módulo criado.**

INFO:port:Protocolo: TCP	INFO:port:Protocolo: UDP
INFO:port:IP origem: 10.0.0.1	INFO:port:IP origem: 10.0.0.1
INFO:port:IP destino: 10.0.0.2	INFO:port:IP destino: 10.0.0.2
INFO:port:MAC origem: 00:00:00:00:00:01	INFO:port:MAC origem: 00:00:00:00:00:01
INFO:port:MAC destino: 00:00:00:00:00:02	INFO:port:MAC destino: 00:00:00:00:00:02
INFO:port:Porta do switch: 1	INFO:port:Porta do switch: 1
INFO:port:VLAN: 0	INFO:port:VLAN: 0
INFO:port:	INFO:port:
IPv4 Packet:	IPv4 Packet:
INFO:port:Protocolo: TCP	INFO:port:Protocolo: UDP
INFO:port:IP origem: 10.0.0.2	INFO:port:IP origem: 10.0.0.2
INFO:port:IP destino: 10.0.0.1	INFO:port:IP destino: 10.0.0.1
INFO:port:MAC origem: 00:00:00:00:00:02	INFO:port:MAC origem: 00:00:00:00:00:02
INFO:port:MAC destino: 00:00:00:00:00:01	INFO:port:MAC destino: 00:00:00:00:00:01
INFO:port:Porta do switch: 2	INFO:port:Porta do switch: 2
INFO:port:VLAN: 0	INFO:port:VLAN: 0

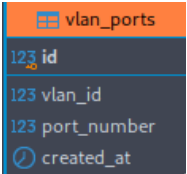
Fonte: Elaborado pelo próprio autor.

Analisando a Figura 10, após realizar o envio de pacotes entre os hosts é possível visualizar que o componente modificado do controlador POX permite a captura dos pacotes que passam pelo switch virtual e a visualização do IP de origem, IP de destino, MAC de origem, MAC de destino, a porta do *switch* e vlan. Ao lado esquerdo da imagem está um fluxo TCP e a direita um fluxo UDP.

### 4.2. Resultados da Criação do Banco de Dados

Foi possível criar o banco de dados utilizando o PostgreSQL. Na Figura 11 é apresentada uma Tabela para armazenar informações das VLANs.

Figura 11 - Tabela para armazenar dados da VLAN.

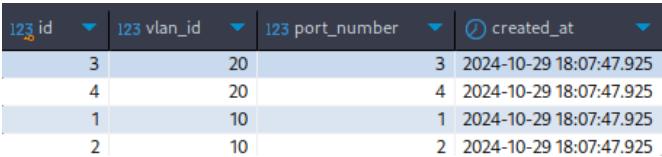


vlan_ports			
123 id			
123 vlan_id			
123 port_number			
123 created_at			

Fonte: Elaborado pelo próprio autor.

Como apresentado na Figura 11, a Tabela com suas respectivas colunas. A coluna ‘vlan\_id’ armazena um valor numérico para determinar qual o identificador da VLAN. ‘port\_number’ armazena a porta do *switch* virtual em que aquela VLAN pertence. ‘created\_at’, é uma coluna para armazenar a data de quando foi criada a VLAN.

Figura 12 - Dados da VLAN inseridos na Tabela.



123 id	123 vlan_id	123 port_number	123 created_at
3	20	3	2024-10-29 18:07:47.925
4	20	4	2024-10-29 18:07:47.925
1	10	1	2024-10-29 18:07:47.925
2	10	2	2024-10-29 18:07:47.925

Fonte: Elaborado pelo próprio autor.

Na Figura 12 apresenta os dados inseridos no banco de dados, para que o controlador POX possa utilizar esses dados nos próximos experimentos.

### 4.3. Resultados da Conexão ao banco de dados com o POX

É realizado o experimento da conexão ao banco de dados utilizando o controlador POX. Como mencionado na seção 3.8 a conexão ao banco de dados é feita a partir de um método chamado ‘load\_vlan\_from\_db’. Esse método além de realizar a conexão a base de dados, é responsável também em consultar os dados para o controlador POX.

Figura 13 - Método utilizado para se conectar a base de dados.

```
def load_vlan_from_db(self, db_config):
    """Load VLAN configuration from PostgreSQL database"""
    vlan_to_ports = {}
    try:
        with psycopg2.connect(**db_config) as conn:
            with conn.cursor() as cursor:
                cursor.execute("SELECT vlan_id, port_number FROM vlan_ports;")
                for vlan_id, port in cursor.fetchall():
                    if vlan_id not in vlan_to_ports:
                        vlan_to_ports[vlan_id] = []
                    vlan_to_ports[vlan_id].append(port)

        log.info("Carregada a configuração da VLAN do banco de dados: %s", vlan_to_ports)
        return vlan_to_ports
    except psycopg2.Error as e:
        log.error("Erro no banco de dados: %s", str(e))
        raise
```

Fonte: Elaborado pelo próprio autor.

Utilizando a biblioteca ‘psycopg2’ foi possível se conectar a base de dados. Essa biblioteca fornece suporte a várias funcionalidades do PostgreSQL. Além de se conectar ao banco de dados, o método apresentado na Figura 13, é responsável por consultar os dados e armazenar em um dicionário, para que o controlador POX possa gerenciar as VLANs. Sendo assim é possível continuar com o restante dos experimentos.

#### 4.4. Experimento - Resultados da Criação das VLANs com o POX

Nesse experimento é realizada a criação das VLANs utilizando o controlador POX. Como apresentado o método ‘load\_vlan\_from\_db’ na Figura 13 é possível observar que além de se conectar a base de dados o método realiza a consulta dos dados. Porém para realizar a gravação da VLAN na tabela de fluxo do switch virtual é necessário utilizar uma mensagem para o switch utilizando o protocolo OpenFlow.

**Figura 14 -Método utilizado para configurar a VLAN no switch.**

```
def setup_vlan(self, event, vlan_id, ports):
    for in_port in ports:
        match = of.ofp_match(in_port=in_port, dl_vlan=vlan_id)
        actions = [of.ofp_action_output(port=of.OFPP_NORMAL)]
        msg = of.ofp_flow_mod(command=of.OFPFC_ADD, priority=100, match=match, actions=actions)
        event.connection.send(msg)

    log.info(f"Configurada a VLAN {vlan_id} no switch {dpid_to_str(event.dpid)} para as portas {ports}")
```

Fonte: Elaborado pelo próprio autor.

Após o POX consultar as VLANs na base de dados, é selecionada as portas do switch virtual e atribuindo a cada uma delas um id da VLAN e criando uma regra de fluxo, como apresentada na Figura 14.

**Figura 15 - Regras de fluxo no switch virtual .**

```
franca1@debian:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=11.427s, table=0, n_packets=0, n_bytes=0, priority=100,in_port="s1-eth3",dl_vlan=20 actions=NORMAL
cookie=0x0, duration=11.386s, table=0, n_packets=0, n_bytes=0, priority=100,in_port="s1-eth4",dl_vlan=20 actions=NORMAL
cookie=0x0, duration=11.386s, table=0, n_packets=0, n_bytes=0, priority=100,in_port="s1-eth1",dl_vlan=10 actions=NORMAL
cookie=0x0, duration=11.386s, table=0, n_packets=0, n_bytes=0, priority=100,in_port="s1-eth2",dl_vlan=10 actions=NORMAL
```

Fonte: Elaborado pelo próprio autor.

Analisando a Figura 15, utilizando o comando ‘ovs-ofctl dump-flows s1’ é possível observar que a regra foi criada com sucesso no switch virtual. Atribuindo a cada porta do switch um VLAN.

**Figura 16 - Inspeccionando o switch virtual.**

```
franca1@debian:~$ sudo ovs-vsctl show
7c4198bd-00a6-4731-bd93-d9ff90120dc2
Bridge s1
  Controller "tcp:127.0.0.1:6633"
    is_connected: true
  fail_mode: secure
  Port s1-eth1
    tag: 10
    Interface s1-eth1
  Port s1
    Interface s1
      type: internal
  Port s1-eth4
    tag: 20
    Interface s1-eth4
  Port s1-eth3
    tag: 20
    Interface s1-eth3
  Port s1-eth2
    tag: 10
    Interface s1-eth2
ovs_version: "3.1.0"
```

Fonte: Elaborado pelo próprio autor.

Após realizar a gravação de uma regra na tabela de fluxo, é possível observar na Figura 16 apresentando as portas utilizadas no switch e a tag com o id da VLAN.

**Figura 17 -Configurando VLANs com o POX.**

```
franca1@debian:~/mininet/pox$ ./pox.py forwarding.l3_learning vlan_controller_db --host=192.1
68.0.124 --dbname=sdn --user=docker --password=docker
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:vlan_controller_db:Loaded VLAN configuration from database: {20: [3, 4], 10: [1, 2]}
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.11.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:vlan_controller_db:Switch 00-00-00-00-00-01 has come up.
INFO:vlan_controller_db:Configured VLAN 20 on switch 00-00-00-00-00-01 for ports [3, 4]
INFO:vlan_controller_db:Configured VLAN 10 on switch 00-00-00-00-00-01 for ports [1, 2]
```

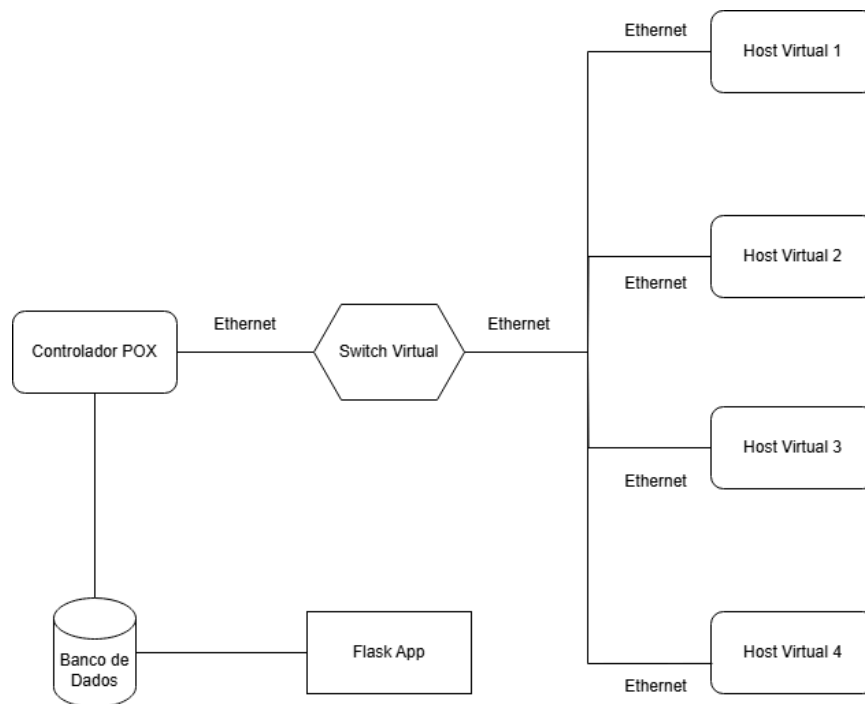
Fonte: Elaborado pelo próprio autor.

Executando o controlador é possível observar que foi atribuída para cada porta selecionada uma VLAN, na Figura 17 é representada na saída do POX que foi configurado as VLANs corretamente no switch.

#### 4.5. Resultados da Criação do protótipo Web

O experimento foi executado, realizando o gerenciamento de VLANs e a utilização do Flask.

**Figura 18 - Topologia Final.**



Fonte: Elaborado pelo próprio autor.

Na Figura 18, a topologia final ficou com um controlador POX, um switch virtual, quatro hosts virtuais, um banco de dados e um app Flask. Com essa topologia já é possível realizar o gerenciamento de VLANs com o protótipo Web.

**Figura 19 - Página principal do protótipo Web.**

## Gerenciar VLANs

Id da VLAN:

Porta do switch:

Enviar

## Lista de VLANs

VLAN ID	Porta do Switch	Ações	
10	1	Editar	Excluir
10	2	Editar	Excluir
20	3	Editar	Excluir
20	4	Editar	Excluir

Fonte: Elaborado pelo próprio autor.



Como pode observar na Figura 19, a página principal contém um formulário para o usuário preencher com as informações necessárias, como, id da VLAN e a porta do switch. Também é apresentada uma Lista com a VLAN associada a uma porta do switch. É possível fazer o gerenciamento das VLANs utilizando os botões de ações, com esses botões é possível editar uma VLAN ou excluí-la.

**Figura 20 - Página principal do protótipo Web.**

## Editar VLAN

Id da VLAN:

Porta do switch:

[Voltar](#)

Fonte: Elaborado pelo próprio autor.

Utilizando o botão de editar, como apresentado na Figura 19, o usuário é direcionado para outra página, observando a Figura 20, onde é possível realizar a edição da VLAN selecionada.

## 5. Considerações finais

Com base nos estudos e experimentos realizados, conclui-se que o gerenciamento de uma rede local virtual é viável utilizando Redes Definidas por Software, pois permite que fluxos de dados sejam separados de acordo com os dados que o controlador é capaz de acessar, como os protocolos TCP e UDP. Os experimentos demonstraram que, através de um sistema web, é possível realizar a administração da rede de maneira remota, simplificando a operação e reduzindo a necessidade de intervenções manuais diretas nos switches. O protótipo desenvolvido mostrou-se funcional ao permitir a execução de funções essenciais para a operação no controlador POX e em switches OpenFlow, sendo possível realizar o gerenciamento de fluxo da rede, independentemente do fabricante do switch ou do firmware utilizado. Recomenda-se, para trabalhos futuros, a incorporação de novas funcionalidades ao projeto. Dentre elas, destaca-se a implementação de uma REST API, adicionalmente, sugere-se a finalização do protótipo web, priorizando a adoção de um design moderno. Essas recomendações visam aprimorar o projeto e ampliar suas possibilidades de aplicação.

## Referências

- CHAGAS, Silvana; LOPES, Nuno; PORTELA, Irene. Performance Evaluation of Host Scalability in Software Defined Networks. 2021 16th Iberian Conference on Information Systems and Technologies (CISTI), Chaves, Portugal, ano 2021, p. 1-6, 12 jul. 2021.
- FERNANDEZ, Carlos; MUNOZ, Jose L. Software Defined Networking (SDN) with OpenFlow 1.3 Open vSwitch and Ryu. UPC Telematics Department, 2015.
- GÖRANSSON, Paul; BLACK, Chuck; CULVER, Timothy. Software Define Networks: A Comprehensive Approach. 2. ed. [S. l.]: Elsevier, 2016. 438 p.
- GRINBERG, Miguel. Flask web development: developing web applications with Python. 2. ed. Sebastopol, CA: O'Reilly Media, 2018.
- HAMEED, A.; WASIM, M. On the Study of SDN for Emulating Virtual LANs. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES (ICICT), 8., 2019, Karachi, Pakistan. Proceedings [...]. Karachi: IEEE, 2019. p. 162-167.
- IEEE. COMPUTER SOCIETY. IEEE standard for local and metropolitan area networks: virtual bridged local area networks. 2005. Disponível em: [http://magrawal.myweb.usf.edu/dcom/Ch3\\_802.1Q-2005.pdf](http://magrawal.myweb.usf.edu/dcom/Ch3_802.1Q-2005.pdf). Acesso em: 2 jun. 2024.
- IPERF. What is iPerf / iPerf3 ?. [S. l.], 2024. Disponível em: <https://iperf.fr/>.
- KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, v. 103, n. 1, p. 14-76, 2015. DOI: 10.1109/JPROC.2014.2371999.
- KREFT, J.; LOFF, G. Mastering PostgreSQL 15: A Comprehensive Guide. [S. l.]: [s. n.], 2022.
- KUROSE, J.; ROSS, K. Redes de computadores e a internet: uma abordagem top-down. 8. ed. [S. l.], 2021. 607 p. ISBN 8582605587.
- MAFIOLETTI, Diego Rossi. Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes. 2015. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Espírito Santo, [S. l.], 2015.
- MCKEOWN, Nick; ANDERSON, Tom; BALAKRISHNAN, Hari; PARULKAR, Guru; PETERSON, Larry; REXFORD, Jennifer; SHENKER, Scott; TURNER, Jonathan. OpenFlow: Enabling innovation in campus networks. Computer Communication Review, [s. l.], p. 69-74, 2008. DOI 10.1145/1355734.1355746.
- MININET. Mininet: An Instant Virtual Network on your Laptop (or other PC).[S. l.], Disponível em: <http://mininet.org/>.
- NADEAU, Thomas D.; GRAY, Ken. SDN: Software Defined Networks. 1. ed. [S. l.]: O'Reilly Media, 2013. 384 p. ISBN 1449342302.

- ONO, D.; IZUMI, S.; ABE, T.; SUGANUMA, T. A Design of Port Scan Detection Method Based on the Characteristics of Packet-In Messages in OpenFlow Networks. In: ASIA-PACIFIC NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (APNOMS), 21., 2020, Daegu, Korea (South). Simpósio [...]. [S. l.: s. n.], 2020. p. 120-125. DOI 10.23919/APNOMS50412.2020.9237012.
- OPENVSWITCH. What is Open vSwitch?. [S. l.], 2024. Disponível em: <https://www.openvswitch.org/>.
- PASSOS, Ana Paula Rocha Soares; BARBOSA, Anderson de Souza; FIGUEIREDO, Eric Reis; FILHO, Marcos Aurélio Constant de Souza; MAIA, Thiago Luis Azevedo; CARVALHO, Yago Meira Lopes de. Software Defined Networks. [S. l.], 2016.
- PRAJAPATI, A.; SAKADASARIYA, A.; PATEL, J. Software defined network: Future of networking. In: INTERNATIONAL CONFERENCE ON INVENTIVE SYSTEMS AND CONTROL (ICISC), 2., 2018, Coimbatore, Índia. Conferência [...]. [S. l.: s. n.], 2018. p. 1351-1354.
- RODRIGUES COSTA, Lucas. OpenFlow e o Paradigma de Redes Definidas por Software. [S. l.], 2013. Disponível em: [https://bdm.unb.br/bitstream/10483/5674/1/2013\\_LucasRodriguesCosta.pdf](https://bdm.unb.br/bitstream/10483/5674/1/2013_LucasRodriguesCosta.pdf). Acesso em: 6 jun. 2024.
- SILVA, Ricardo Anísio da. Otimização de tráfego broadcast em redes Openflow. 2017. Dissertação (Mestre em Ciência da Computação) - Universidade Federal de Pernambuco, Recife, 2017.
- SEGMENTAÇÃO de LANs e VLANs. [S. l.], 17 jun. 2020. Disponível em: [https://wiki.sj.ifsc.edu.br/index.php/IER60808:\\_Segmenta%C3%A7%C3%A3o\\_de\\_LANs\\_e\\_VLANs](https://wiki.sj.ifsc.edu.br/index.php/IER60808:_Segmenta%C3%A7%C3%A3o_de_LANs_e_VLANs).
- TANENBAUM, A. S.; FEAMSTER, N.; WETHERALL, D. J. Redes de Computadores. 6. ed. [S. l.]: Bookman, 2021. 624 p. ISBN 978-8582605608.
- VIRTUALBOX. Welcome to VirtualBox.org!. [S. l.], 2024. Disponível em: <https://www.virtualbox.org/>.