CONSTRUÇÃO DE SWITCH OPENFLOW UTILIZANDO DISPOSITIVO RASPBERRY PI 3.0

Trabalho de Conclusão de Curso

Engenharia da Computação

Eduardo Castilho de Souza Silva
Orientador: Edison de Queiroz Albuquerque

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

Eduardo Castilho de Souza Silva

CONSTRUÇÃO DE SWITCH OPENFLOW UTILIZANDO DISPOSITIVO RASPBERRY PI 3.0

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, dezembro de 2019.

Eduardo Castilho de Souza Silva
Construção de Switch OpenFlow utilizando dispositivo Raspberry Pi 3.0/ Eduardo Castilho de Souza Silva. – Recife - PE, Brasil, dezembro de 2019-
64 p.
Orientador: Prof. Edison de Queiroz Albuquerque

Trabalho de Conclusão de Curso – Engenharia de Computação
Escola Politécnica de Pernambuco
Universidade de Pernambuco, dezembro de 2019.

1. Redes Definidas por Software. 2. O protocolo OpenFlow. 3. A plataforma Open vSwitch. 4. A arquitetura Raspberry Pi. 5. *Framework:* Ryu *Controller*. I. Prof. Edison de Queiroz Albuquerque. II. Universidade de Pernambuco. III. Escola Politécnica de Pernambuco.

Este trabalho é dedicado à minha família.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 16/12/2019, às 10h, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **EDUARDO CASTILHO DE SOUZA SILVA**, orientado(a) pelo(a) professor(a) **EDISON DE QUEIROZ ALBUQUERQUE**, sob título Construção de Switch OpenFlow Utilizando Dispositivo Raspberry Pi 3.0, a banca composta pelos professores:

SÉRGIO MURILO MACIEL FERNANDES (PRESIDENTE)
EDISON DE QUEIROZ ALBUQUERQUE (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 10 (DEZ)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá 7 dias para entrega da versão final da monografia a contar da data deste documento.

Sérgio Murilo Maciel Fernandes

AVALIADOR 1: Prof (a) **SÉRGIO MURILO MACIEL FERNANDES**

Edison de Queiroz Albuquerque

AVALIADOR 2: Prof (a) **EDISON DE QUEIROZ ALBUQUERQUE**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Dedico a todos que em algum momento participaram em minha jornada.

Agradecimentos

Primeiramente agradeço e dedico este trabalho a minha Mãe Glória, ao meu Pai Gilberto e ao meu irmão Gilberto Junior, os quais não cabem em palavras tamanho apoio e suporte que proporcionaram para esta minha jornada.

Dedico este trabalho também a toda minha família principalmente aqueles em memória que nunca deixaram de acreditar em meu potencial para passar por essa fase importante de minha vida.

Agradeço muito aos meu amigos e colegas de faculdade por todos os trabalhos e estudos que fizemos juntos e espero encontra-los pelos caminhos da vida. Também agradeço aos Professores da POLI por todas as lições passadas a mim durante todo o curso.

Finalmente agradeço ao meu orientador, Professor Edison, por todo o conhecimento passados a mim durante suas disciplinas e por toda sabedoria demostrada pela sua orientação durante a realização deste trabalho.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Autorização de publicação de PFC

Eu, **Eduardo Castilho de Souza Silva** autor(a) do projeto de final de curso intitulado: **Construção de Switch OpenFlow Utilizando Dispositivo Raspberry Pi 3.0;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Eduardo Castilho de Souza Silva

Eduardo Castilho de Souza Silva

Orientador(a): **Edison de Queiroz Albuquerque**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves**

Data: 16/12/2019

Resumo

As redes de computadores foram concebidas como um meio de compartilhar dispositivos periféricos, como impressoras, *modems* e vários outros, existindo em ambientes acadêmicos e governamentais, assim como em empresas de grande porte. Porém os paradigmas tradicionais de redes de computadores apresentam um alto custo de alteração de *hardware* e estarão, em um futuro próximo, no seu limite quanto a utilização em plataformas de: computação em nuvem, virtualização e nos *data centers*. O paradigma de Redes Definidas por Software, do inglês *Software Defined Networks* (SDN), uma tecnologia derivada do princípio de redes programáveis surgiu para cumprir os requisitos modernos de Qualidade de Serviço das redes de computadores, do inglês QoS. O SDN permite a separação entre os planos de controle e de dados, tornando assim possível o desenvolvimento de *switches* e roteadores e encaminhando dados de acordo com as regras definidas por um controlador e instaladas no *switch*. Entretanto os *switches* SDN disponíveis no mercado possuem um custo elevado e são feitos para projetos de grande porte, dificultando o desenvolvimento de aplicações controladoras e a implantação dessas em redes SDN de pequeno porte ou ainda a nível de simulação. Este trabalho apresentará uma maneira de montar um *switch* SDN a um custo aproximado de R$ 300,00 utilizando o dispositivo Raspberry Pi 3.0, o protocolo OpenFlow, o mais popular para implementação de SDN e a tecnologia que virtualiza esse protocolo a nível de software, denominada OpenvSwitch. Foram executadas aplicações controladoras SDN na rede criada para comprovar o funcionamento do *switch*.

Palavras-chave: Raspberry Pi. SDN. Switch. OpenFlow. Open vSwitch. Controlador SDN.

Abstract

Computer networks were designed as a means of sharing peripheral devices such as printers, modems, and many others, existing in academic and government environments, as well as in large enterprises. But traditional computer networking paradigms have a high cost of hardware change and will be in the near future at their limit for use on cloud computing, virtualization, and data center platforms. Software Defined Networks (SDN) paradigm, a technology derived from the programmable networks principle has emerged to meet modern QoS computer network Quality of Service requirements. SDN enables separation between control and data plans and it becomes possible to develop switches and routers routing data according to the rules defined by a controller and installed on the switch. However, commercially available SDN switches are costly and are designed for large projects, making it difficult to develop controller applications and deploy them on small or simulated SDN networks. This paper will present a way to build an SDN switch at an approximated cost of R$ 300.00 using the Raspberry Pi 3.0 device, the OpenFlow protocol, the most popular for SDN implementation, and the technology that virtualizes this software-level protocol, called OpenvSwitch. SDN controller applications were run on the network created to verify the switch's operation.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

A-CPI – Application-Controller Plane Interface

API – Application Programming Interface

CPU – Central Processing Unit

D-CPI – Data-Controller Plane Interface

GB – Giga Byte

GCC – GNU Compiler Collection

GHz – Giga-Hertz

GNU – GNU is Not Unix

GPIO – General Purpose Input/Output

HDMI – High-Definition Multimedia Interface

IEEE – Institute of Eletrical and Eletronics Engineers

IP – Internet Protocol

IPv4 – Internet Protocol version 4

IPv6 – Internet Protocol version 6

IRTF – Internet Research Task Force

LAN – Local Area Network

MAC – Media Access Control

MB – Mega Byte

Mpbs – Mega bits per second

MHz – Mega-Hertz

ONF – Open Network Foundation

OSI – Open System Interconnection

OSS – Operations Support System

PC – Personal Computer

QoS – Quality of Service

RAM – Random Access Memory

SD – Secure Digital

SDN – Software Defined Network

SoC – System on Chip

SSH – Secure Shell

TCP – Transmission Control Protocol

TLS – Transport Layer Security

USB – Universal Serial Bus

Capítulo 1

Introdução

“Uma rede SDN é a separação física do plano de controle de rede do plano de encaminhamento e onde um plano de controle controla vários dispositivos” [1]. “Essa separação permite que ambos os planos evoluam de forma independente e traz inúmeras vantagens, como alta flexibilidade, sendo independente do fornecedor, programável e possibilidade de realizar uma visão centralizada da rede” [2].

“O protocolo OpenFlow é um elemento fundamental para a construção de soluções SDN” [1]. “O OpenFlow é baseado em um *switch Ethernet*, com uma tabela de fluxo interna e uma interface padronizada para adicionar e remover entradas de fluxo” [4]. “O OpenFlow fornece um protocolo aberto para programar a tabela de fluxo em diferentes *switches* e roteadores. Um administrador de rede pode particionar o tráfego em fluxos de produção e pesquisa. Os pesquisadores podem controlar seus próprios fluxos - escolhendo as rotas que seus pacotes seguem e o processamento que recebem. Dessa maneira, os pesquisadores podem tentar novos protocolos de roteamento, modelos de segurança, esquemas de endereçamento e até alternativas ao IP. Na mesma rede, o tráfego de produção é isolado e processado da mesma maneira que hoje” [4].

“Como alternativa à compra de um *switch* de *hardware* caro com o *firmware* habilitado para OpenFlow, um grande número de dispositivos diferentes pode ser transformado em um *switch* compatível com OpenFlow através de implementações de software” [5].

“O Open vSwitch é um *switch* de software multicamada licenciado sob a licença Apache 2 de código aberto. Nosso objetivo é implementar uma plataforma de *switch* de qualidade de produção que suporte *interfaces* de gerenciamento padrão e abra as funções de encaminhamento para controle e extensão programática” [3].

“O Raspberry Pi é um computador de "placa única", ou seja, é um computador pequeno, para que você possa carregar um sistema operacional básico e usá-lo como um PC de baixa potência” [6].

1.1 Objetivos do Trabalho

O objetivo geral deste projeto é construir um *switch* SDN – *Software Defined Network* (Redes Definidas por *Software*) com o dispositivo Raspberry Pi 3.0, para se obter um baixo custo de montagem da rede e criar uma plataforma de roteamento para o desenvolvimento, simulação e implantação de aplicações de controlador SDN.

Outros objetivos específicos a serem alcançados são:

- Automatizar o funcionamento do *switch* e execução das aplicações controladoras da rede;
- Analisar o funcionamento do protocolo OpenFlow e da arquitetura da plataforma construída;
- Criar um *switch* que possua um número maior de portas para maior capacidade de conexão da rede;
- Demonstrar o funcionamento de aplicações controladoras com funcionalidades diversas;

1.2 Metodologia e estratégia de ação

Para alcançar os objetivos geral e específicos do trabalho, a metodologia aplicada consistirá de pesquisar a literatura de trabalhos científicos como os apresentados em [2] e [4], relacionados a: redes SDN; OpenFlow; Open vSwitch; arquitetura Raspberry Pi; Sistema Operacional Linux e suas distribuições em *sites*, *blogs*, *e-books* e outros meios oriundos de fontes relevantes à área dos temas pesquisados.

Será feito um estudo descritivo do processo de construção da plataforma, listando e descrevendo a escolha dos componentes tanto de *hardwar*e quanto de *software*, sequência de instalações e comandos necessários, montagem da estrutura física

com fotos referentes a cada componente utilizado. Posteriormente será realizado simulações de aplicações SDN já desenvolvidas na plataforma construída. Por fim será feita uma análise dos resultados obtidos.

1.3 Resultados e Impactos Esperados

Ao final do trabalho espera-se que a plataforma construída esteja plenamente funcional de acordo com os objetivos propostos e que este trabalho sirva de guia para o estudo de redes SDN com a criação de *switches* do protocolo OpenFlow, implementando o software Open vSwitch na arquitetura Raspberry Pi 3.0 ou em outras arquiteturas que suportarem a plataforma.

1.4 Organização do Trabalho

Além deste capítulo, esta monografia está estruturada da seguinte forma:

- Capítulo 2 – Abordará as definições de Redes Definidas por Software, segundo, várias fontes, bem como os conceitos dos elementos que a compõe e também abordará o protocolo que implementa as redes SDN, o OpenFlow e a tecnologia que virtualiza esse protocolo a nível de *software*, o OpenvSwitch, e sobre esses irá mostrar os seus conceitos e componentes.

- Capítulo 3 – Apresentará o processo de construção do dispositivo *switch* SDN, primeiramente abordando os componentes utilizados, posteriormente demonstrando os passos e comandos realizados para a instalação e o funcionamento correto do dispositivo e por fim apresentará a configuração do ambiente de testes planejado, descrevendo a estrutura da rede.

- Capítulo 4 – Apresentará, no ambiente de testes em funcionamento, as aplicações controladoras SDN que serão utilizadas bem como o

framework utilizado para a programação das mesmas, nesse caso o RYU *Controller*. Além disso discutirá os resultados obtidos nos testes realizados e usará métricas de comparação com outras configurações de *switch* SDN.

- Capítulo 5 – Conclusão: Apresentará a conclusão do trabalho, considerações finais e trabalhos futuros.

Capítulo 2

Redes Definidas por *Software*

As Redes Definidas por *Software* (SDN) é uma tecnologia proveniente do conceito de redes programáveis, cuja a ideia é de ser capaz de: inicializar, controlar, modificar e gerenciar o comportamento da rede dinamicamente via interfaces abertas. O elemento principal que essa tecnologia por conceito traz é o de separar o plano de controle do plano de roteamento da rede, e possibilitar aplicações que possuam os meios necessários para programar o controle da rede estruturada.

A origem da SDN vem da instituição IRTF onde o tipo de estrutura da rede idealizado foi o fator determinante para sua concepção [5]. E em 2011 surgiu a ONF para promover o uso de SDN e OpenFlow. Este capítulo apresentará as definições de entidades e pesquisadores sobre o assunto e apresentará o modelo de implementação de SDN: OpenFlow, sua estrutura e características e apresentará o OpenvSwitch, uma tecnologia que implementa o protocolo OpenFlow a nível de *software*, seu funcionamento e seus componentes.

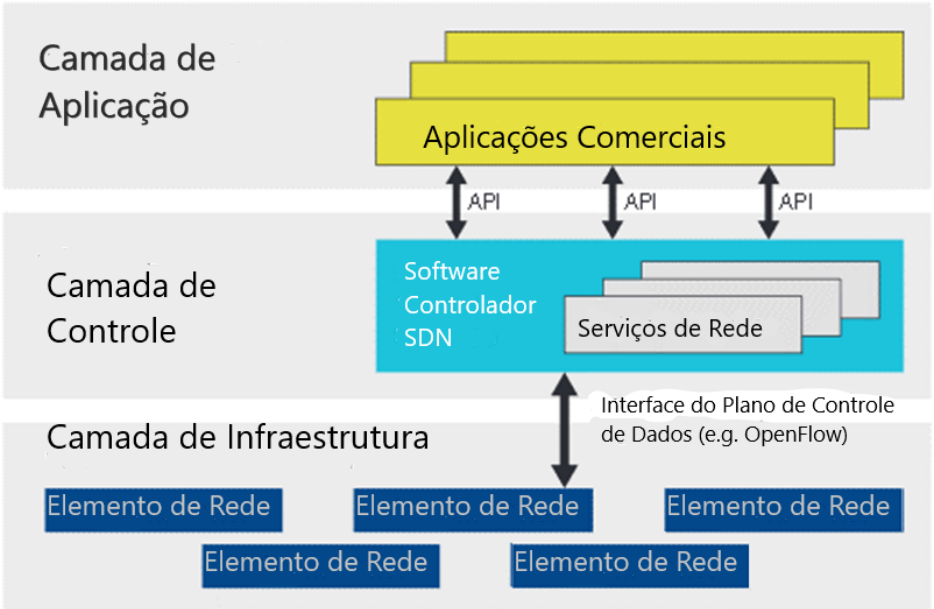

Figura 1. Arquitetura SDN [9]

2.1 Conceitos Básicos

A definição de alguns conceitos básicos relacionados as redes de computadores será necessária para uma melhor compreensão do tema proposto.

Linux Kernel: "O Kernel do Linux é um programa que constitui o núcleo central de um sistema operacional de computador. Possui controle completo sobre tudo o que ocorre no sistema" [11].

"O kernel pode ser contrastado com um *shell* (como bash, csh ou ksh em sistemas operacionais semelhantes ao UNIX), que é a parte mais externa de um sistema operacional e um programa que interage com os comandos do usuário. O próprio kernel não interage diretamente com o usuário, mas interage com o *shell* e outros programas, bem como com os dispositivos de *hardware* no sistema, incluindo o processador (também chamado de unidade central de processamento ou CPU), memória e unidades de disco" [11].

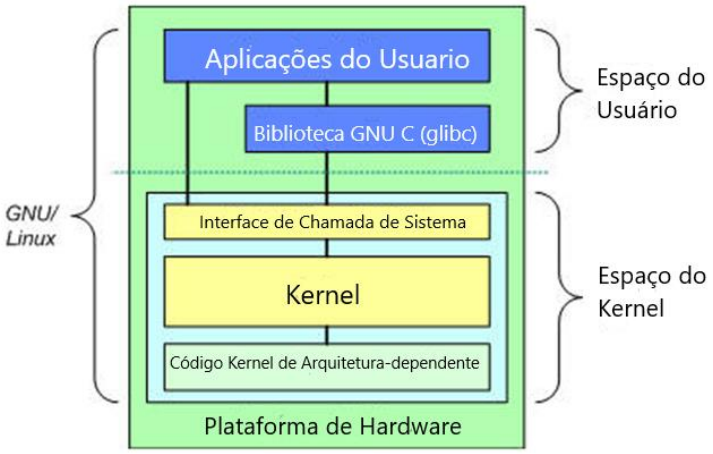

Figura 2. Estrutura de Blocos do Linux [15].

De acordo com a Figura 2, [15] afirma que "no topo está o espaço do usuário, ou aplicativo. É aqui que os aplicativos do usuário são executados. Abaixo do espaço do usuário está o espaço do kernel. Aqui, o kernel do Linux existe".

Switch: Os *switches* são dispositivos da camada de enlace (podem ser da camada de rede) do modelo OSI que podem conectar duas ou mais redes. Os *switches* são os únicos dispositivos que permitem micro segmentação. Os *switches* de corte são mais rápidos porque, quando um pacote chega, ele o encaminha logo depois de olhar apenas para o endereço de destino. Um *switch* de armazenamento e

encaminhamento inspeciona o pacote inteiro antes de encaminhar. A maioria dos comutadores não pode parar o tráfego de *broadcast*. Os *switches* são considerados dispositivos de *link* de dados dedicados porque estão próximos a 100% da largura de banda. Um *switch* pode ser pensado como uma ponte de várias portas" [16].

Firewall: "Um *firewall* é um dispositivo de segurança de rede que monitora o tráfego de entrada e saída da rede e decide se deve permitir ou bloquear o tráfego específico com base em um conjunto definido de regras de segurança. Os *firewalls* são a primeira linha de defesa em segurança de rede há mais de 25 anos. Eles estabelecem uma barreira entre redes internas seguras e controladas, que podem ser confiáveis, das redes externas não confiáveis, como a Internet. Um *firewall* pode ser *hardware*, *software* ou ambos" [17].

Controlador SDN: "uma entidade de *software* que possui controle exclusivo sobre um conjunto de recursos do plano de dados. Um controlador SDN também pode oferecer uma instância de modelo de informações abstraídas para pelo menos um cliente" [8].

"Um controlador SDN pode ser implementado com qualquer número de componentes de *software*, que residem em qualquer número de plataformas físicas. Os componentes distribuídos são necessários para manter uma visão sincronizada e auto consistente das informações e do estado. Esse requisito limita o conceito de um único controlador SDN; componentes de *software* que não compartilham essa característica são necessariamente externos ao controlador" [8].

NE (*Network Element* – Elemento de Rede): "Um grupo de recursos do plano de dados que são gerenciados como uma única entidade. Um elemento de rede (NE) fornece um espaço de nome comum usado pelo controlador SDN para acessar recursos que encaminham, manipulam ou armazenam dados do usuário" [8].

NFV (*Network Functions Virtualization* – Virtualização de Funções de Rede): "A NFV é a execução de funções de rede tradicionais, normalmente em uma máquina virtual executada em *hardware* comum. Isso permite que as operadoras usem sua infraestrutura de servidor x86 para executar serviços de rede que costumavam ser executados em placas de serviço em roteadores ou *switches* ou em *hardware* especializado" [7].

CPI (*Controller Plane Interface* – *Interface* de plano controlador): “a *interface* genérica para uma instância do modelo de informações SDN. Uma instância de CPI pode ser mais especializada com um caractere de prefixo” [8].

A-CPI (*Application-Controller Plane Interface* – *Interface* do plano controlador com o plano de aplicações): “Uma *interface* entre aplicações e o controlador SDN cuja aplicação recebe serviços do controlador SDN” [8].

D-CPI (Data-*Controller Plane Interface* – *Interface* de plano controlador com o plano de dados): “Uma *interface* entre dados e os planos controladores, cujo controlador SDN controla os recursos do plano de dados” [8].

OSS (*Operations Support System* – Sistema de Suporte a Operações): “As funções de gerenciamento podem ser executadas por qualquer número de entidades, cujos detalhes estão fora do escopo da arquitetura SDN” [8].

Packet (Pacote): “um quadro Ethernet, incluindo cabeçalho e carga útil” [10].

Port (Porta): “onde os pacotes entram e saem do *pipeline* do OpenFlow. Pode ser uma porta física, uma porta lógica definida pelo *switch* ou uma porta reservada definida pelo protocolo OpenFlow” [10].

Flow Table (Tabela de fluxo): “um estágio do *pipeline*, contém entradas de fluxo” [10].

Flow Entry (Entrada de fluxo): “um elemento em uma tabela de fluxo usado para combinar e processar pacotes. Ele contém um conjunto de campos para correspondências de pacotes, um conjunto de contadores para rastrear pacotes e um conjunto de instruções a serem aplicadas” [10].

Pipeline: “o conjunto de tabelas de fluxo vinculadas que fornecem correspondência, encaminhamento e modificações de pacotes em um comutador OpenFlow” [10].

Match Field (Campo de correspondência): “um campo no qual um pacote é comparado, incluindo cabeçalhos de pacotes, a porta de entrada e o valor de metadados” [10].

Instruction (Instrução): “uma operação que tanto contém um conjunto de ações a serem adicionadas como contém uma lista de ações a serem aplicadas imediatamente ao pacote ou modificar o processamento do *pipeline*” [10].

Action (Ação): “uma operação que encaminha o pacote para uma porta ou modifica o pacote. As ações podem ser especificadas como parte do conjunto de instruções associado a uma entrada de fluxo ou em um bloco de ações associado a uma entrada de grupo. ” [10].

Action Set (Conjunto de Ações): “Um conjunto de ações associadas ao pacote que são acumulados enquanto o pacote está sendo processado por cada tabela e que são executados quando o conjunto de instruções instrui o pacote a sair do *pipeline* de processamento” [10].

Action Bucket (Balde de Ações): “um conjunto de ações e parâmetros associados, definidos para grupos” [10].

Group (Grupo): “uma lista de *Action Bucket* e alguns meios de escolher um ou mais desses baldes para aplicar mudanças a um nível de pacote” [10].

Counters (contadores): “são métricas de tempo do OpenFlow mantidas por cada tabela, fluxo, porta, fila, grupo e balde de ações. Contadores compatíveis com OpenFlow pode ser implementado em *software* e mantido por contadores de *hardware* com intervalos mais limitados” [10].

2.2 Definições de SDN

Existem um grande número de definições utilizadas no ambiente de redes de computadores entre institutos, industrias e universidades para descrever o que são as SDN. Alguns as equivalem simplesmente ao protocolo OpenFlow, outros definem que são simplesmente a separação do plano de controle da rede do plano de roteamento e há os que definam que são redes “programáveis”. Dentre essas podem ser citadas:

“As SDN são habilitadoras que permitem uma função de orquestração de serviços de ponta a ponta na rede. A função Orquestração interage com os controladores SDN que fornecem o componente de rede dessa orquestração. O controlador SDN configura o encaminhamento de fluxos dentro de uma rede e os direciona para locais diferentes, dependendo dos requisitos do orquestrador” [7].

"O objetivo da SDN é fornecer *interfaces* abertas que permitam o desenvolvimento de *software* que possa controlar a conectividade fornecida por um conjunto de recursos de rede e o fluxo de tráfego de rede por eles, juntamente com a possível inspeção e modificação do tráfego que pode ser executado na rede" [8].

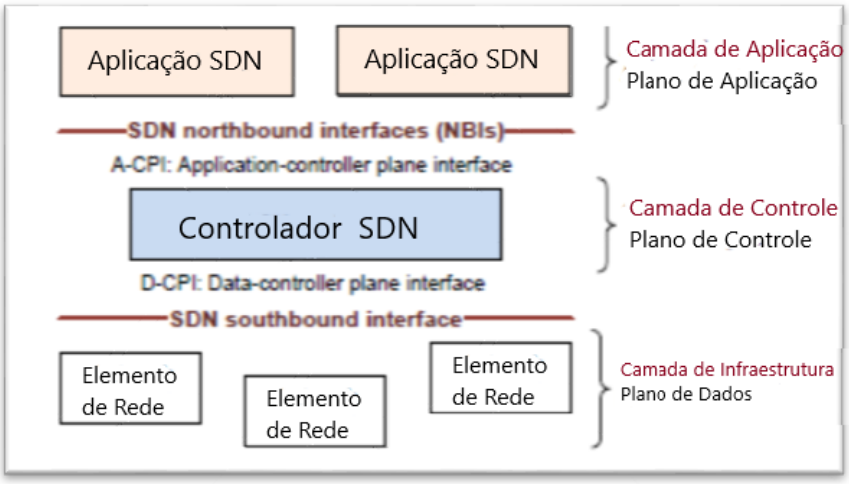

Figura 3. Componentes Básicos SDN [8].

"As aplicações SDN existem no plano de aplicação e comunica seus requisitos de rede ao plano controlador via *northbound interfaces*, geralmente chamadas NBIs. No meio, o controlador SDN converte requisitos das aplicações e exerce controle de baixo nível sobre os elementos da rede, enquanto fornecendo informações relevantes até as aplicações SDN. Um controlador SDN pode orquestrar demandas de aplicações concorrentes por recursos de rede limitados, de acordo com a política da rede" [8].

Além da NBIs existem as SBIs, *southbound interfaces,* que permitem a comunicação do controlador com os NE (elementos de rede) para realização de configurações e mudanças de forma dinâmica na rede. O protocolo OpenFlow atua nessa *interface*.

Sendo assim os princípios das SDN podem ser resumidos em:

- Desacoplamento do controlador dos planos de dados

  "Este princípio exige controladores separáveis e planos de dados. No entanto, entende-se que o controle deve necessariamente ser exercido dentro dos sistemas de plano de dados. O D-CPI (*Data-Controller Plane Interface*) entre o controlador SDN e o elemento de rede é definido de forma que o controlador

SDN possa delegar funcionalidade significativa ao NE, mantendo-se ciente do estado do NE" [8].

- Controle logicamente centralizado

  "Em comparação com o controle local, um controlador centralizado tem uma perspectiva mais ampla dos recursos sob seu controle e pode potencialmente tomar melhores decisões sobre como implantá-los. A escalabilidade é aprimorada dissociando e centralizando o controle, permitindo visualizações cada vez mais globais, mas menos detalhadas, dos recursos da rede" [8].

- Exposição de recursos de rede abstratos e estado para aplicativos externos

Outros conceitos importantes a serem definidos no que se refere a SDN são apresentados a seguir:

2.2.1 O Plano de Dados

"O plano de dados compreende um conjunto de um ou mais elementos de rede, cada um dos quais contém um conjunto de recursos de encaminhamento ou processamento de tráfego. Os recursos são sempre informações de uma entidade ou de entidades físicas subjacentes" [8].

2.2.2 O Plano Controlador

"O plano do controlador compreende um conjunto de controladores SDN, cada um dos quais tem controle exclusivo sobre um conjunto de recursos expostos por um ou mais elementos de rede no plano de dados (seu período de controle) " [8].

"A funcionalidade mínima do controlador SDN é executar fielmente as solicitações das aplicações suportadas, enquanto isola cada aplicação de todas as outras. Para executar esta função, um controlador SDN pode se comunicar com controladores SDN de mesmo nível, controladores SDN subordinados ou ambientes não SDN, conforme necessário" [8].

"Uma função comum, mas não essencial, de um controlador SDN é atuar como o elemento de controle em um *loop* de *feedback*, respondendo a eventos de rede para

se recuperar de falhas, otimizar novamente as alocações de recursos ou outros fatores" [8].

2.2.3 O Plano de Aplicações

"O plano de aplicações compreende um ou mais aplicativos, cada um dos quais possui controle exclusivo de um conjunto de recursos expostos por um ou mais controladores SDN. *Interfaces* adicionais para aplicativos não são excluídas. Uma aplicação pode chamar ou colaborar com outras aplicações. Uma aplicação pode atuar como um controlador SDN por si só" [8].

2.2.4 Gerenciamento de SDN

"Cada aplicação controladora SDN e elemento de rede possui uma *interface* funcional para um gerente da rede" [8].

"A funcionalidade mínima do gerente é alocar recursos de uma mistura de recursos no plano inferior para uma entidade cliente específica no plano superior e estabelecer informações de acessibilidade que permitam que as entidades do plano inferior e superior se comuniquem mutuamente" [8].

"A funcionalidade de gerenciamento adicional não é impedida, sujeita à restrição de que a aplicação controladora SDN ou o elemento de rede possuam controle exclusivo sobre qualquer recurso" [8].

2.2.5 O Coordenador SDN

Para configurar os ambientes cliente e servidor, é necessária a funcionalidade de gerenciamento. O coordenador é o componente funcional do controlador SDN que atua em nome do gerente. Clientes e servidores exigem gerenciamento, em todas as perspectivas sobre modelos de dados, controle e plano de aplicativos.

2.2.6 O Agente SDN

Qualquer protocolo deve terminar em algum tipo de entidade funcional. Um modelo controlador-agente é apropriado para a relação entre uma entidade controlada e uma entidade controladora e se aplica recursivamente à arquitetura SDN. A entidade controlada é designada agente, um componente funcional que representa os recursos e as capacidades do cliente no ambiente do servidor.

"Um agente em um determinado controlador SDN no nível **N** de abstração, representa os recursos e ações disponíveis para um cliente ou aplicação do controlador SDN, no nível mais alto do modelo. Um agente no plano de dados do nível mais baixo de abstração, representa os recursos e ações disponíveis para o controlador SDN do nível **N** especificado" [8].

2.2.7 A arquitetura SDN

Todos os componentes anteriormente citados são reunidos na arquitetura para implementação de SDNs. A Figura 4 ilustra a relação entre esses componentes:

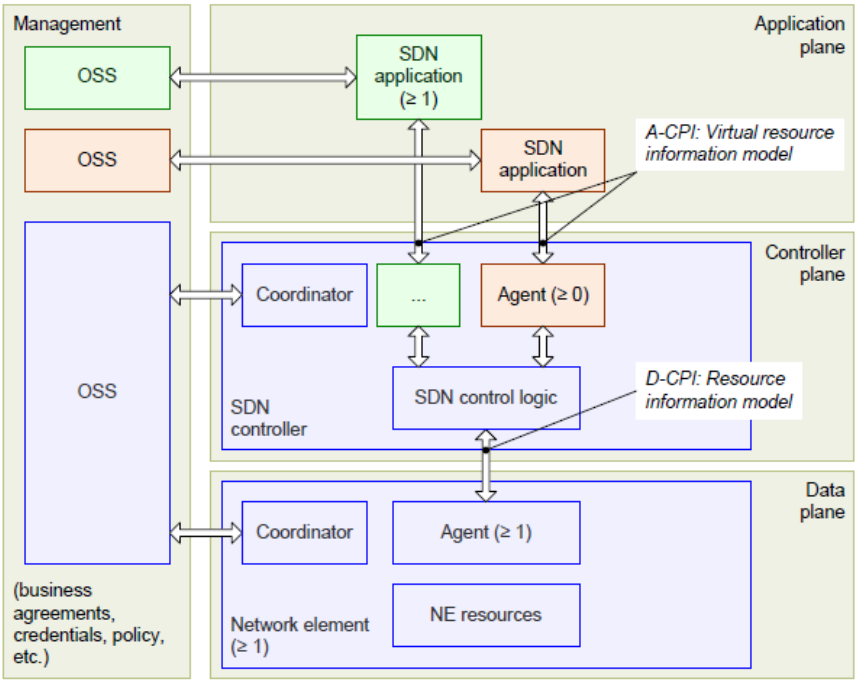

Figura 4. Visão geral da arquitetura SDN [8].

2.3 O protocolo OpenFlow

Segundo [10] "o canal OpenFlow é a *interface* que conecta cada *switch* OpenFlow a um controlador. Através desse canal, o controlador configura e gerencia o *switch*, recebe eventos e envia pacotes em alguma porta do *switch"*.

Ainda segundo [10], "Entre o caminho de dados e o canal OpenFlow, a *interface* é específica da implementação, no entanto todas as mensagens do canal OpenFlow

devem ser formatadas de acordo com o protocolo OpenFlow. O canal geralmente é criptografado usando TLS, mas pode ser executado diretamente sobre TCP".

"O protocolo OpenFlow suporta três tipos de mensagens: controlador a *switch*, assíncrono e simétrico, cada um com vários subtipos. As mensagens do controlador para o *switch* são iniciadas pelo controlador e usadas para gerenciar ou inspecionar diretamente o estado do *switch*. Mensagens assíncronas são iniciadas pelo *switch* e usadas para atualizar o controlador de eventos de rede e alterações no estado do *switch*. Mensagens simétricas são iniciado pelo *switch* ou pelo controlador e enviado sem solicitação. Os tipos de mensagens usados pelo protocolo OpenFlow são descritos a seguir: " [10].

2.3.1 Mensagens Controlador para Switch

As mensagens do controlador para o *switch* são iniciadas pelo controlador e podem ou não exigir uma resposta do *switch*, conforme a seguir:

Features: O controlador pode solicitar os recursos de um *switch* enviando uma solicitação de recursos. O *switch* deve responder com uma resposta de *Features* que especifique os recursos do *switch*. Isso é comumente realizado após o estabelecimento do canal OpenFlow.

Configuration: O controlador pode definir e consultar parâmetros de configuração no *switch*. O *switch* responde apenas a uma consulta vinda do controlador.

Modify-State: as mensagens de *Modify-State* são enviadas pelo controlador para gerenciar o estado nos *switches*. Seu objetivo principal é adicionar, excluir e modificar entradas de fluxo / grupo nas tabelas do OpenFlow e definir propriedades da porta do *switch*.

Read-State: as mensagens de *Read-State* são usadas pelo controlador para coletar estatísticas do *switch*.

Packet-out: estas são usados pelo controlador para enviar pacotes de uma porta especificada no *switch*, e encaminhar pacotes recebidos via mensagens de *Packet-in*. As mensagens de *Packet-out* devem conter um pacote completo ou um ID de *buffer* que faz referência a um pacote armazenado no *switch*. A mensagem também

deve conter uma lista de ações a serem aplicadas na ordem em que são especificadas. Uma lista de ações vazia descarta o pacote.

Barrier: as mensagens de *request / reply* de barreira são usadas pelo controlador para garantir-se as dependências da mensagem foram atendidas ou para receber notificações de operações concluídas.

2.3.2 Mensagens Assíncronas

Mensagens assíncronas são enviadas sem um controlador solicitando-as de um *switch*. Os *switches* enviam mensagens assíncronas para os controladores para indicar a chegada de pacotes, a alteração de estado do *switch* ou o erro. Os principais tipos de mensagens assíncronas são descritos a seguir:

Packet-in: transfere o controle de um pacote para o controlador. Para todos os pacotes que não possuam uma entrada de fluxo correspondente, um evento de entrada de pacote pode ser enviado ao controlador, dependendo da configuração da tabela. Para todos os pacotes encaminhados para a porta reservada do controlador, um evento de entrada de pacote é sempre enviado para controladores.

Flow-Removed: informa ao controlador sobre a remoção de uma entrada de fluxo de uma tabela de fluxo. As mensagens *Flow-removed* são enviadas apenas para fluxo com o conjunto de sinalizadores OFPFF_SEND_FLOW_REM. Eles são gerados como o resultado de uma solicitação de exclusão do fluxo do controlador ou do processo de expiração do fluxo do comutador quando um dos tempos limite do fluxo é excedido.

Port-status: informa o controlador de uma alteração em uma porta. Espera-se que o *switch* envie as mensagens *Port-status* para os controladores à medida que a configuração da porta ou o estado da porta são alterados. Esses eventos incluem mudança de eventos de configuração de porta, por exemplo, se ele foi derrubado diretamente por um usuário e alteração do estado da porta eventos, por exemplo, se o *link* foi desativado.

Error: o *switch* pode notificar os controladores sobre problemas usando mensagens de erro.

2.3.3 Mensagens Síncronas

Mensagens simétricas são enviadas sem solicitação, em qualquer direção, as quais são descritas abaixo:

Hello: as mensagens de *Hello* são trocadas entre o *switch* e o controlador na inicialização da conexão.

Echo: as mensagens de *request / reply* de *Echo* podem ser enviadas do switch ou do controlador e devem retornar um *Echo reply*. Eles são usados principalmente para verificar a vitalidade de uma conexão entre o *switch* e o controlador e podem também pode ser usado para medir sua latência ou largura de banda.

Experimenter: as mensagens do *experimenter* fornecem uma maneira padrão de os switches OpenFlow oferecerem funcionalidade no espaço de tipo de mensagem OpenFlow. Esta é uma área de preparação para os recursos para futuras revisões do OpenFlow.

2.4 Componentes do Switch OpenFlow

"Um OpenFlow *switch* consiste em uma ou mais tabelas de fluxo e tabela de grupo, que executam pesquisas e encaminhamento de pacotes e um canal de fluxo aberto para um controlador externo. O *switch* se comunica com o controlador e o controlador gerencia o *switch* por meio do protocolo OpenFlow. Usando o protocolo OpenFlow, o controlador pode adicionar, atualizar e excluir entradas de fluxo nas tabelas de fluxo, tanto reativamente (em resposta a pacotes) quanto proativamente. Cada tabela de fluxo no *switch* contém um conjunto de entradas de fluxo, cada entrada de fluxo consiste em campos, contadores e um conjunto de instruções para aplicar aos pacotes correspondentes" [10].

Alem disso segundo [4] "Um *switch* OpenFlow consiste em pelo menos três partes:

- Uma tabela de fluxo, com uma ação associada a cada fluxo, para dizer ao *switch* como processar o fluxo.
- Um Canal que conecta o *switch* a um processo remoto (chamado controlador), permitindo comandos e pacotes sejam enviados entre um controlador e o *switch*.

- O protocolo OpenFlow, que fornece uma maneira aberta e padronizada para um controlador se comunicar com um *switch*".

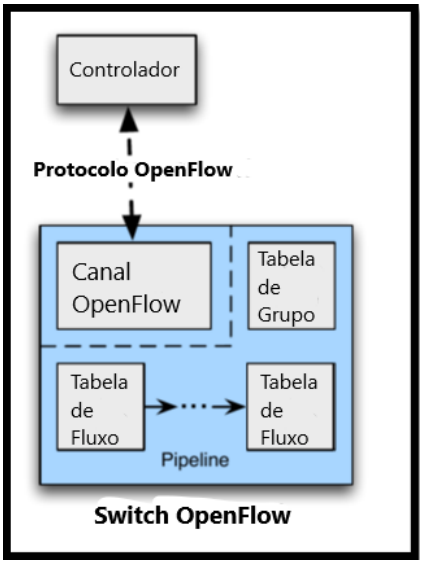

Figura 5. Arquitetura do *Switch* OpenFlow [20].

2.4.1 Portas OpenFlow

Segundo [10] a porta OpenFlow é a interface de rede para passar pacotes entre o processamento OpenFlow e o restante da rede. Os *switches* OpenFlow se conectam logicamente entre si através de suas portas OpenFlow.

"O *switch* OpenFlow disponibiliza várias portas para processamento OpenFlow. O conjunto da porta OpenFlow pode não ser idêntico ao conjunto de interfaces de rede fornecidas pelo hardware do *switch*, alguma *interface* de rede pode estar desabilitada para o OpenFlow, e o *switch* OpenFlow pode definir Portas OpenFlow" [10].

"Pacotes OpenFlow são recebidos em uma porta de entrada, processada pelo *pipeline* OpenFlow que pode encaminhá-los para uma porta de saída" [10].

Ainda segundo [10], "o *switch* OpenFlow possui três tipos de portas OpenFlow, **portas físicas**, **portas lógicas** e **portas reservadas**".

As **portas físicas** do OpenFlow são portas definidas por *switch* que correspondem a uma *interface* de *hardware* do *switch*. Por exemplo, em um *switch Ethernet*, as portas físicas mapeiam de um para um para as *interfaces Ethernet*.

As **portas lógicas** do OpenFlow são portas definidas por *switch* que não correspondem diretamente a um *hardware interface* do *switch*. Portas lógicas são

abstrações de nível superior que podem ser definidas no *switch* usando métodos não-OpenFlow.

As **portas reservadas** especificam ações de encaminhamento genérico como enviar para o controlador, *flood* (inundar) ou encaminhar um pacote utilizando métodos não OpenFlow, como por exemplo um processamento “normal” de um *switch*.

São **portas reservadas**:

- **ALL:** Representa todas as portas que o *switch* pode usar para encaminhar um pacote específico.
- **CONTROLLER:** Represente o canal de controle com o controlador OpenFlow. Pode ser usado como uma porta de entrada ou como uma porta de saída.
- **TABLE**: representa o início do *pipelin*e do OpenFlow.
- **IN_PORT:** Representa a porta de entrada de pacotes. Pode ser usado apenas como uma porta de saída, enviando o pacote pela mesma porta de entrada.
- **ANY:** Valor especial usado em alguns comandos do OpenFlow quando nenhuma porta é especificada.
- **LOCAL:** Representa a pilha de rede local do *switch*. Pode ser usado como uma porta de entrada ou como uma porta de saída. A porta local permite que entidades remotas interajam com o *switch* via OpenFlow na rede, e não através de uma rede de controle separada.
- **NORMAL:** Representa o *pipeline* não-OpenFlow tradicional do comutador.
- **FLOOD:** Representa inundações usando a tubulação normal do *switch.* Pode ser usado somente como uma porta de saída, em geral, enviando o pacote para todas as portas padrão, mas não para a porta de entrada.

2.4.2 Tabela de Fluxo

Segundo [10], uma tabela de Fluxo de um *switch* OpenFlow consiste em:

Tabela 1. Componentes Principais de uma Tabela de Fluxo [10].

Match Fields	Counters	Instructions

Cada entrada da tabela de fluxo é identificada por seus campos de correspondência e contém:

- ***Match Fields*:** para combinar com pacotes. Eles consistem na porta de entrada e nos cabeçalhos de pacotes, e opcionalmente metadados especificados por uma tabela anterior.
- ***Counters*:** atualizar os valores para pacotes correspondentes.
- ***Instructions*:** para modificar o conjunto de ações ou processamento de *pipeline*.

De acordo com [21], cada entrada na tabela de fluxo do *switch* tem uma ação simples associada. As três ações básicas são:

- *Foward* (Encaminhamento) dos pacotes desse fluxo para uma determinada porta (ou portas). Isso permite que os pacotes sejam roteados pela rede. Na maioria dos *switches*, espera-se que isso ocorra à taxa de linha.
- Encapsular e encaminhar os pacotes desse fluxo para um controlador. O pacote é entregue ao *secure channel* (canal seguro), onde é encapsulado e enviado para um controlador.
- *Drop* (Descarte) os pacotes desse fluxo. Pode ser usado para segurança, para conter ataques de negação de serviço ou para reduzir o tráfego suspeito de descoberta de *broadcast* (transmissão) dos *host*s finais.

O cabeçalho do fluxo é uma tupla de 10 campos. Cada campo de cabeçalho pode ser um *wildcard* (curinga) para permitir a agregação de fluxos. Alguns exemplos de entradas e ações da tabela de fluxo são mostrados abaixo:

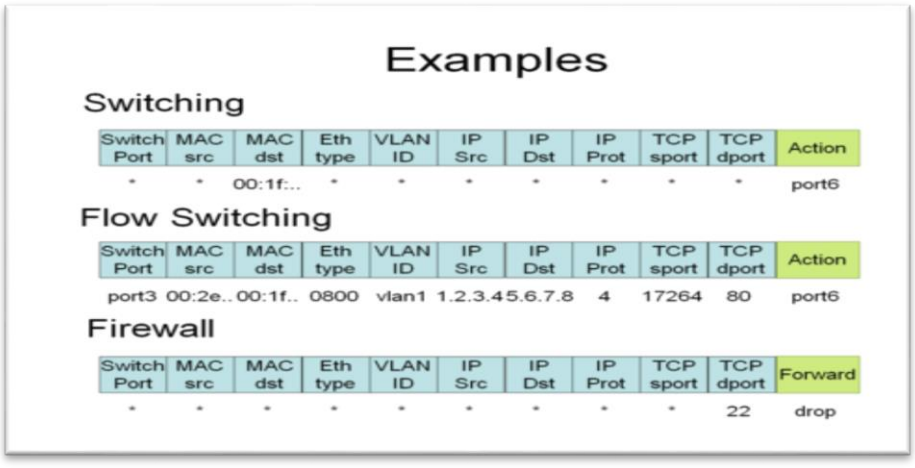

Figura 6. Exemplos de entradas da tabela de fluxos [21].

Na Figura 7 é ilustrada o *pipeline* de processamento de um pacote OpenFlow, onde os pacotes que entram são comparados com várias tabelas de fluxo e podem, ou serem descartados ou ter o seu conjunto de ações executado.

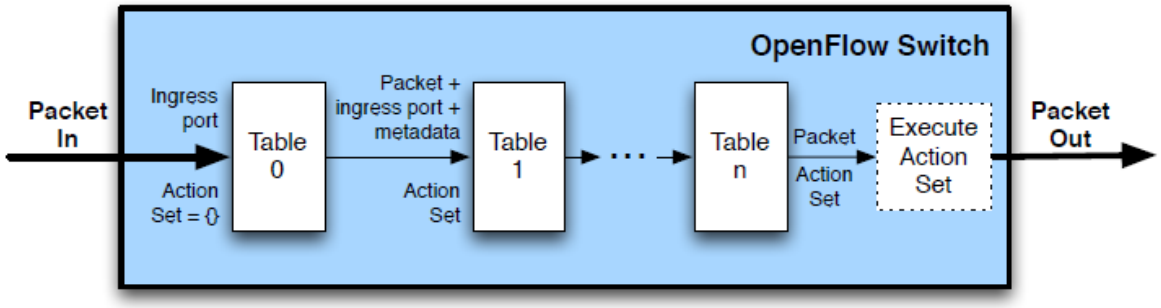

Figura 7. O pipeline de processamento de um pacote OpenFlow [10].

2.4.3 Tabela de Grupos

Uma tabela de grupo consiste em entradas de grupo na rede. A capacidade de um fluxo apontar para um grupo permite ao OpenFlow representar métodos adicionais de encaminhamento.

Tabela 2. Componentes Principais de uma Tabela de Grupo [10]

Group Identifier	Group Type	Counters	Action Buckets

Cada linha de grupo é identificada pela coluna ***Group Identifier*** e contém:

- ***Group Identifier*****:** um número inteiro sem sinal de 32 bits que identifica exclusivamente o grupo.
- ***Counters*****:** atualizados quando os pacotes são processados por um grupo.
- ***Action Buckets*****:** uma lista ordenada de ações, em que cada intervalo de ações contém um conjunto de ações a executar e parâmetros associados.

2.4.4 Correspondência de Pacotes

O *switch* começa executando uma pesquisa de tabela na primeira tabela de fluxo e, com base no processamento de *pipeline*, pode executar uma consulta de tabela em outras tabelas de fluxo como é ilustrado na Figura 8.

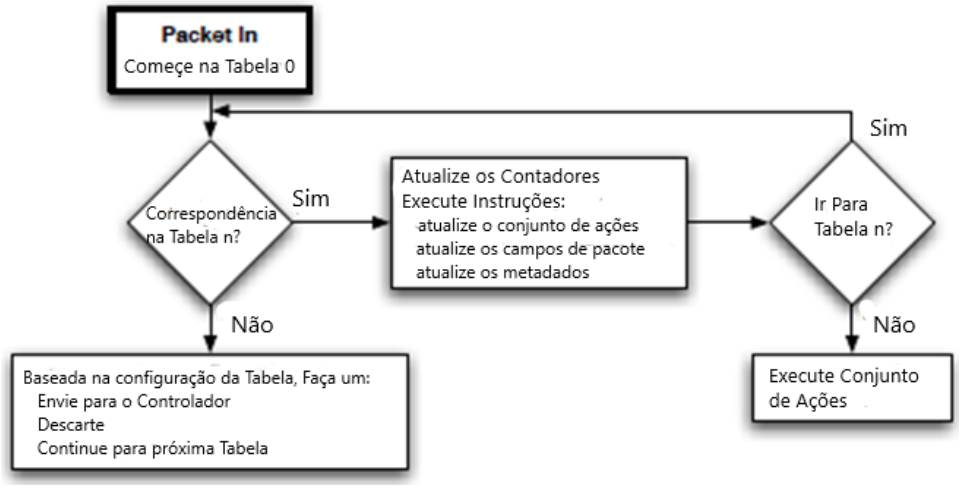

Figura 8. Fluxograma de um pacote em um *switch* OpenFlow [10].

Os campos de correspondência de pacotes usados para pesquisas de tabela dependem no tipo de pacote e geralmente incluem vários campos de cabeçalho de pacote, como endereço de origem *Ethernet* ou endereço de destino IPv4. Além dos cabeçalhos de pacotes, as comparações também podem ser realizadas com a porta de entrada e os campos de metadados. Os metadados podem ser usados para passar informações entre tabelas em uma troca.

2.5 A plataforma Open vSwitch

O Open vSwitch é comumente usado como um *switch* SDN, e a principal maneira de controlar o encaminhamento é o OpenFlow. Através de um protocolo binário simples, o OpenFlow permite ao controlador adicionar, remover, atualizar, monitorar e obter estatísticas sobre tabelas de fluxo e seus fluxos, bem como desviar pacotes selecionados para o controlador e injetar pacotes do controlador para o *switch*.

“No Open vSwitch, o *daemon switch* recebe os fluxos OpenFlow de um Controlador SDN correspondente aos pacotes recebidos do módulo de *datapath* nessas tabelas OpenFlow, reúne as ações a serem aplicadas e, finalmente, armazena em *cache* o resultado no *datapath* do kernel. Isso permite que o módulo de *datapath* permaneça desconhecendo os detalhes do protocolo OpenFlow, simplificando-o ainda mais. A partir do ponto de vista do controlador OpenFlow, o armazenamento em *cache* e a separação em componentes de usuário e kernel são detalhes invisíveis à implementação. Na visão do controlador, cada pacote visita uma série de tabelas de

fluxo OpenFlow e o *switch* encontra a prioridade mais alta no fluxo cujas condições são satisfeitas pelo pacote, e executa suas ações OpenFlow" [3].

2.5.1 Componentes da distribuição

O conjunto de componentes principais possui as seguintes descrições para as distribuições às quais a plataforma pode ser instalada:

- **ovs-vswitchd**: um *daemon* que implementa o *switch*, junto com um módulo de kernel Linux para roteamento baseado em fluxo.
- **ovsdb-server**: um servidor de banco de dados leve que o ovs-vswitchd consulta para obter sua configuração.
- **ovs-dpctl**: uma ferramenta para configurar o módulo do kernel do *switch*.
- **ovs-vsctl**: um utilitário para consultar e atualizar a configuração do ovs-vswitchd.
- **ovs-appctl**: um utilitário que envia comandos para a execução de *daemons* do Open vSwitch.

O Open vSwitch também provisiona algumas ferramentas auxiliares:

- **ovs-ofctl**: um utilitário para consultar e controlar *switches* OpenFlow.
- **ovs-pki**: um utilitário para criar e gerenciar a infraestrutura de chave pública dos *switches* OpenFlow.
- **ovs-testcontroller**: um simples controlador OpenFlow que pode ser útil para testes (embora não seja para produção).

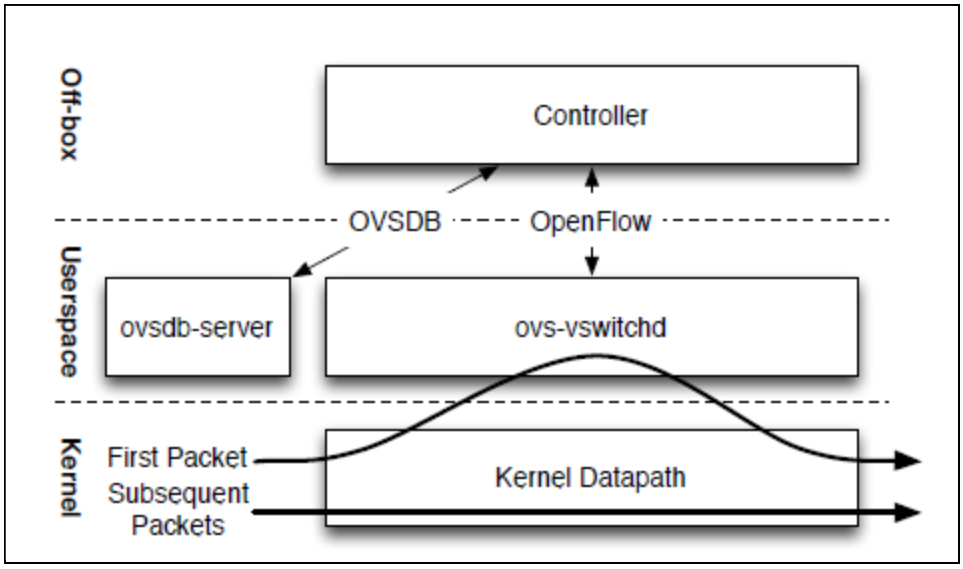

Figura 9. Diagrama componentes Open vSwitch [3].

A Figura 9 ilustra os componentes e *interfaces* do Open vSwitch. O primeiro pacote de um fluxo resulta em uma falha e o módulo do kernel direciona o pacote para o componente *userspace*, que armazena em *cache* a decisão de encaminhamento para pacotes subsequentes no kernel.

2.6 Considerações

As SDN juntamente com as suas implementações tanto físicas quanto virtuais são muito versáteis para lidar com os requisitos modernos de QoS como disponibilidade, escalabilidade, entre outros.

Neste capítulo foram vistas as definições teóricas e conceituais sobre as SDN, OpenFlow e Open vSwitch, assim como os seus respectivos componentes para um melhor esclarecimento sobre a estrutura de rede a ser montada.

Capítulo 3

Construção do Dispositivo

Neste capítulo aborda-se o processo de construção, o qual inicia-se pela explicação sobre a arquitetura Raspberry Pi depois lista-se todos os componentes a serem utilizados na construção. A seguir descreve-se o modo como os componentes são instalados (caso necessário) e por fim explica-se todos os passos necessários para configurar uma rede SDN na plataforma instalada.

3.1 A arquitetura Raspberry Pi

"Raspberry Pi é o nome de uma série de computadores de placa única fabricados pela Raspberry Pi *Foundation*, uma instituição de caridade do Reino Unido que visa educar as pessoas em computação e criar um acesso mais fácil à educação em computação" [18].

"O Raspberry Pi foi lançado em 2012 e houve várias iterações e variações lançadas desde então" [18].

"O Pi original possui uma CPU de 700 MHz de núcleo único e apenas 256 MB de RAM, e o modelo mais recente possui uma CPU de 1,4 GHz de quatro núcleos com 1 GB de RAM. O principal preço para o Raspberry Pi sempre foi US $ 35 e todos os modelos custam US $ 35 ou menos, incluindo o Pi Zero, que custa apenas US $ 5" [18].

A Tabela 3 demonstra os respectivos modelos de Raspberry Pi e suas configurações:

Tabela 3. Tipos de modelos Raspberry Pi [6]

	Raspberry Pi 3 Model B	**Raspberry Pi 2 Model B**	**Raspberry Pi Model B+**	**Raspberry Pi Model A+**	**Raspberry Pi Zero & Zero W**
Processor Chipset	*Broadcom BCM2837 64-bit quad-core processor*	*Broadcom BCM2837 64-bit quad core processor*	*Broadcom BCM2835 32-bit single-core processor*	*Broadcom BCM2835 32-bit single-core processor*	*Broadcom BCM2835 32-bit single-core processor*
Processor Speed	*1.2 GHz*	*900 MHz*	*700 Mhz*	*700 Mhz*	*1 GHz*
RAM	*1 GB*	*1 GB*	*512 MB*	*256 MB*	*512 MB*
Storage	*MicroSD*	*MicroSD*	*MicroSD*	*MicroSD*	*MicroSD*
Output	*4 USB ports*	*4 USB ports*	*4 USB ports*	*1 USB port*	*1 micro-USB port*
GPIO	*40-pin*	*40-pin*	*40-pin*	*40-pin*	*40-pin*
Ethernet Port	*Yes*	*Yes*	*Yes*	*No*	*No*
Wi-Fi	*Yes*	*No*	*No*	*No*	*No (Zero)/Yes (Zero W)*
Bluetooth	*Yes*	*No*	*No*	*No*	*No (Zero)/Yes (Zero W)*

Quanto a especificação vista em [19] do modelo utilizado no trabalho (Raspberry Pi 3 modelo B+), apresenta os seguintes componentes:

- **Processador:** Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz
- **Memória:** 1GB LPDDR2 SDRAM
- **Conectividade**:
    - 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac *wireless LAN, Bluetooth* 4.2
    - *Gigabit Ethernet over* USB 2.0 (*maximum throughput* 300 Mbps)
    - 4 × USB 2.0 *ports*
- **Acesso:** *Extended 40-pin* GPIO *header*
- **Som e Vídeo:**

- 1 × *full size* HDMI
- MIPI DSI *display port*
- MIPI CSI *camera port*
- 4 *pole stereo output and composite video port*

- **Multimedia:** H.264, MPEG-4 *decode (1080p30); H.264 encode(1080p30); OpenGL ES 1.1, 2.0 graphics*
- **Armazenamento**: Micro SD *format for loading operating system and data storage*

Quanto a organização, os componentes da arquitetura Raspberry Pi do modelo utilizado são ilustrados pela Figura 10:

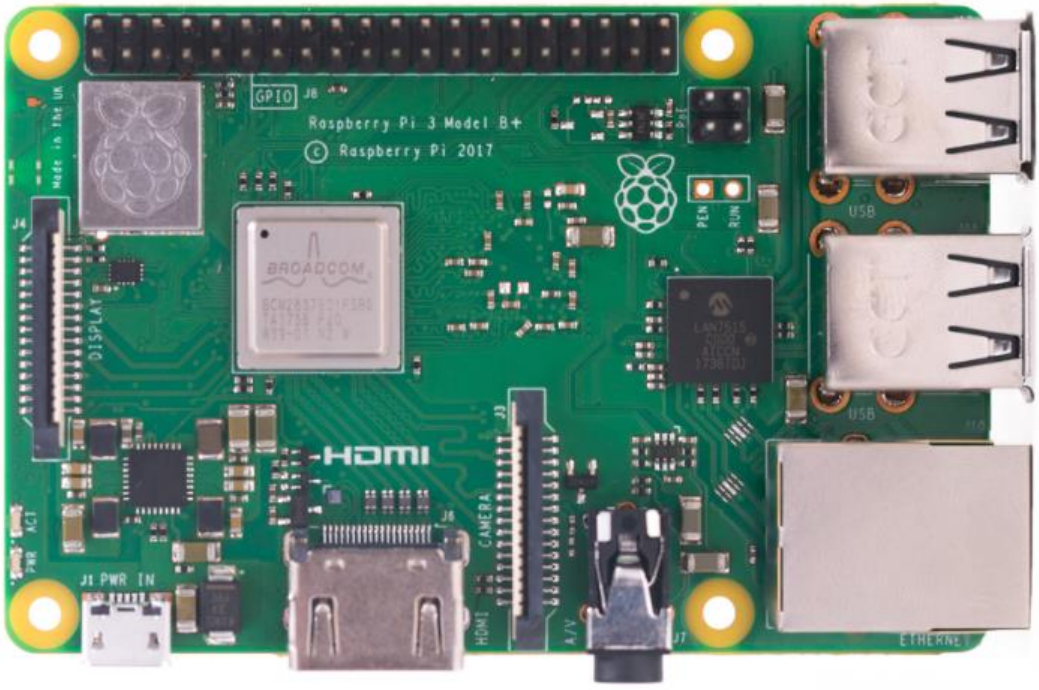

Figura 10. Raspberry Pi 3 Model B+ [19].

3.2 Componentes

Para o processo de construção do dispositivo fez-se necessário a utilização de componentes de *hardware* e *software,* os quais são listados abaixo:

Hardware:

- Placa Raspberry Pi 3.0 *model* B+;
- USB 2.0 *Ethernet Adapter*;
- Cartão de Memória *Micro* SD 16 GB com adaptador;
- Cabos *Ethernet*;
- Teclado e *Mouse*;

Figura 11. Kit Raspberry Pi 3.0 utilizado como base.

Software:

- Raspbian Buster *with desktop* Kernel 4.19.75;
- Open vSwitch versão 2.12.0;
- Ryu *Controller*
- Win32 *Disk Imager*;
- A Ferramenta Git;
- Wireshark;

3.3 O Processo de Instalação

3.3.1 A instalação do Raspbian Linux

Para possuir um sistema operacional que forneça suporte para as plataformas de SDN mencionadas fez-se como escolha a instalação do Raspbian Linux, o qual é fornecido pela própria organização que mantêm a arquitetura Raspbery Pi e está disponível no seguinte *link:*

https://www.raspberrypi.org/downloads/raspbian/

A versão escolhida foi a ***Raspbian Buster with desktop,*** a versão mais recentemente lançada, possuindo a versão de kernel do Linux: **4.19.**

Baixado a imagem para instalação em outro computador, foi instalado o *software* **Win32 Disk Imager,** o qual é utilizado para escrever a imagem binária no cartão *micro* SD através do adaptador para cartão SD.

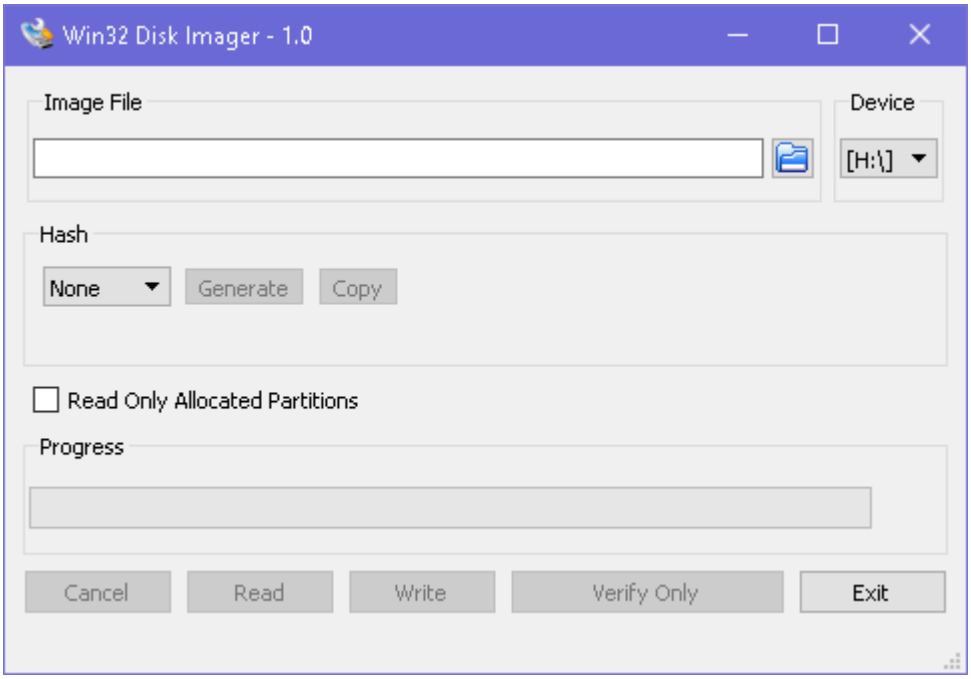

Figura 12. *Software* Win32 Disk Imager [Autor].

Uma vez escrita a imagem no cartão de memória procede-se o primeiro uso do Raspbian, onde o processo de instalação realmente se inicia.

No processo de instalação visual, escolhe-se as opções de linguagem, *download* de pacotes, formatação do cartão SD e outros elementos que ficam a critério do usuário

além das credenciais de acesso por padrão que são o usuário: **pi** e a senha: **raspberry.**

3.3.2 Requisitos para a instalação do Open vSwitch

Os requisitos para a instalação do Open vSwitch segundo [13] são:

- O comando “GNU make”;
- Um compilador C, com:
  - GCC 4.6 ou posterior.
  - Clang 3.4 ou posterior.
- Python 3.4 ou posterior. Deve-se também possuir a biblioteca Python6 versão 1.4.0 ou posterior.
- No Linux, módulo do kernel que acompanha o Linux 3.3 e posterior. Os recursos e o desempenho abertos do vSwitch podem variar com base no módulo e no kernel. (na instalação a única versão do Open vSwitch compatível com o módulo kernel (4.19.75) Instalado no Raspberry Pi foi a versão 2.12.10.

3.3.3 A instalação do Open vSwitch

O processo para instalar a plataforma Open vSwitch em uma distribuição Linux consiste em primeiro possuir uma conexão com internet e possuir a ferramenta Git (esta ferramenta é nativa para a distribuição Linux instalada no Raspberry).

Segundo [14] “O Git é um sistema de controle de versão de código distribuído rápido, escalável e com um conjunto de comandos bastante rico que fornece operações de alto nível e acesso total aos seus mecanismos internos. ” Para o escopo do trabalho o único comando que será necessário do conjunto do Git será:

git clone https://github.com/openvswitch/ovs.git

Ainda segundo [14] “O git *clone,* clona um repositório em um novo diretório”, esse repositório no caso é o conjunto de arquivos necessários para a instalação do Open vSwitch.

Após isso é necessário descompactar o repositório baixado e entrar no diretório criado pela descompactação, para isso utiliza-se os comandos:

tar -xvzf openvswitch -2.12.0. tar.gz

cd openvswitch -2.12.0

Antes de iniciar o processo de compilação, é necessário a instalação e atualização de dependências de compilação e a partir deste ponto os comandos são executados a nível de *super* usuário do Linux (o que equivale ao Administrador do Windows):

sudo su

apt -get update

apt -get install python - simplejson python -qt4 libssl –dev python - twisted - conch automake autoconf gcc uml - utilities libtool build - essential pkg- config

Para configurar o pacote executando o *script configure*, geralmente pode-se chamar o configure sem argumentos no diretório descompactado do Open vSwitch. Por exemplo:

./configure

Por padrão, todos os arquivos são instalados no caminho **/usr/local**. O Open vSwitch também espera encontrar seu banco de dados em **/usr/local/etc/openvswitch** por padrão.

Para construir o módulo do kernel do Linux, executando o *switch* baseado em kernel, é necessário passar o local do diretório de construção do kernel com a opção: **--with-linux**. Por exemplo, para construir para uma instância em execução do Linux o comando utilizado é:

./configure --with-linux=/lib/modules/$(uname -r)/build

Onde o comando **uname -r** irá obter a versão do kernel do sistema Linux instalado para o *hardware* em questão.

Para compilar e instalar os executáveis no sistema em execução, por padrão, em **/usr/local**, é necessário rodar o seguinte comandos GNU:

make

make install

Para carregar os módulos do kernel instalados executa-se o comando:

/sbin/modprobe openvswitch

E para verificar se os módulos estão carregados e se o módulo *openvswitch* está listado, no caminho **/sbin/,** executam-se os comandos em conjunto:

lsmod | grep openvswitch

O comando **grep** procura por padrões em diretórios de arquivos correspondentes ao parâmetro passado e o comando **lsmod** lista os módulos carregados, como ilustrado na Figura 13:

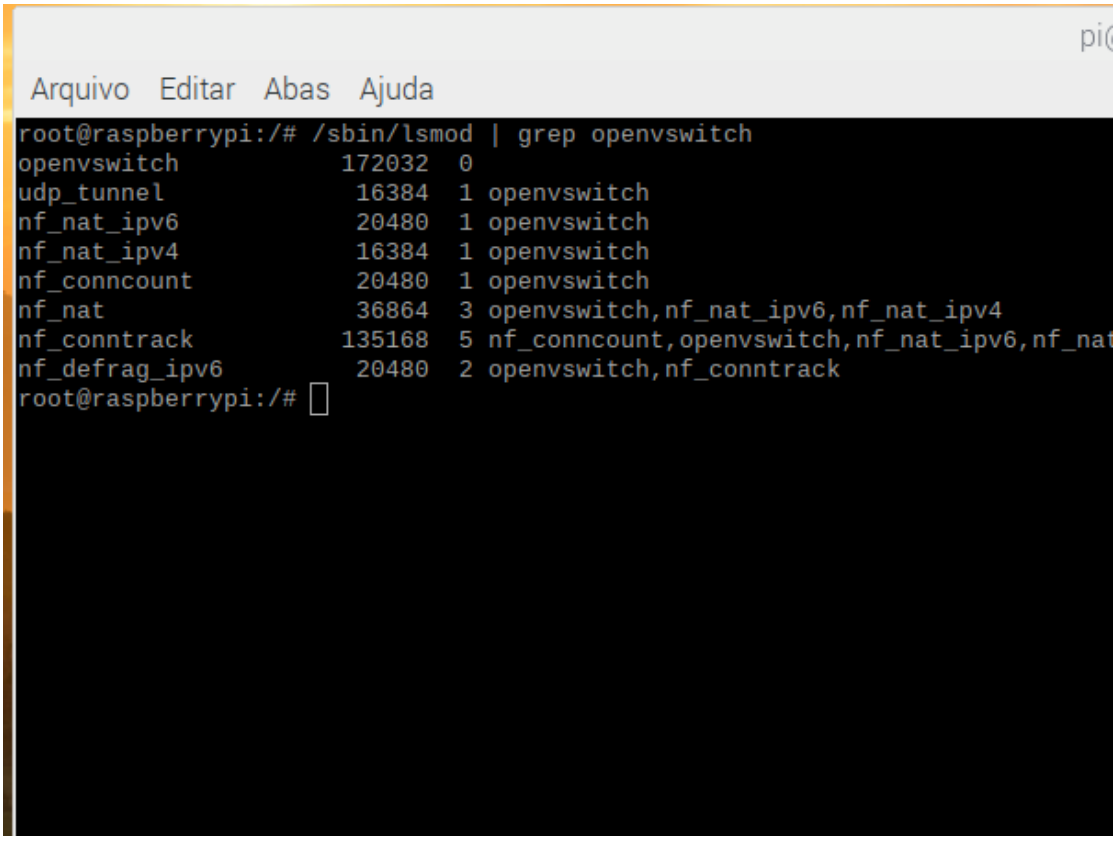

Figura 13. Lista dos módulos Linux instalados para execução do Open vSwitch [Autor].

Para iniciar o Open vSwitch, é necessário carregar os componentes (vistos na seção inicial do Open vSwitch) nas variáveis de ambiente do sistema operacional. A partir disso pode-se iniciar o programa através do utilitário **ovs-ctl,** onde o processo padrão instalação fica localizado no caminho: **/usr/local/share/openvswitch/scripts,** o qual é necessário passar como argumento da variável de ambiente PATH do Linux para estar disponível na linha de comando do sistema. Os comandos são:

export PATH=$PATH:/usr/local/share/openvswitch/scripts

ovs-ctl start

Após isso o Open vSwitch deve inicializar corretamente no sistema, como ilustrado na Figura 14:

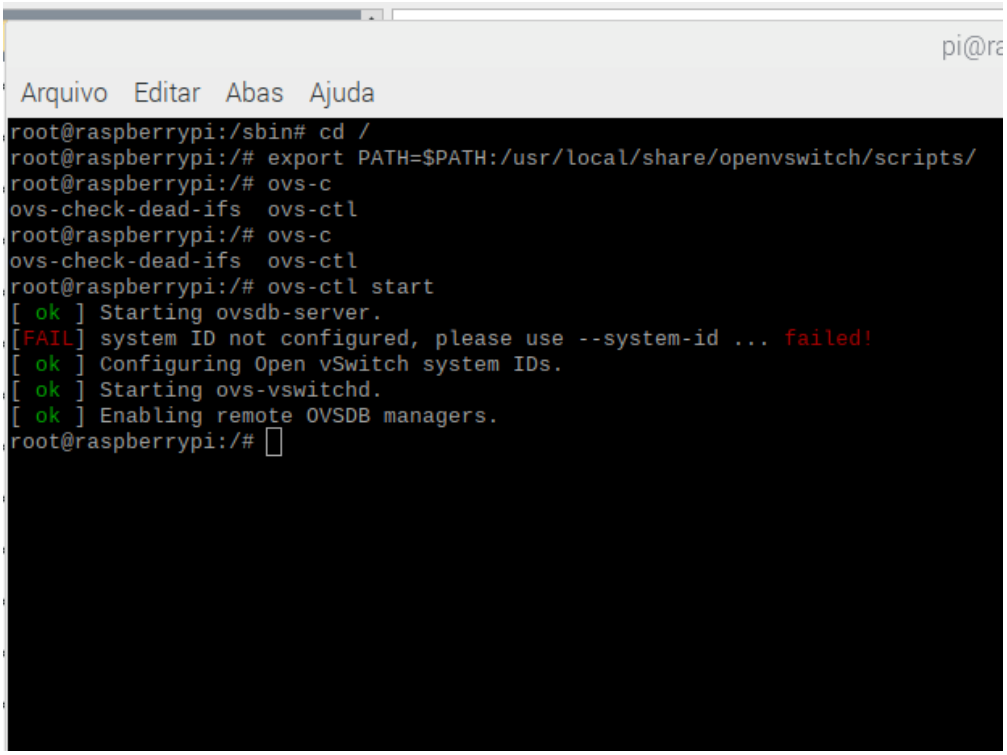

Figura 14. Inicialização do Open vSwitch [Autor].

Para deixar o conjunto de comandos do Open vSwitch sem precisar utilizar o comando **export**, é necessário injetar o caminho de diretórios *scripts* do Open vSwitch no arquivo **.bashrc** no diretório **~** do Raspbian, isso é feito com os seguintes comandos:

cd ~

nano .bashrc

E desse modo insere-se no texto do arquivo o comando **export** mencionado.

3.4 Configuração do Ambiente de Simulação

3.4.1 A configuração do Open vSwitch

Para a configuração do ambiente de simulação, faz-se necessário pensar sobre que tipo de rede pretende-se instalar. Nesse caso a nível de teste, idealizou-se uma rede com o *switch* SDN, um Controlador SDN interno e dois *hosts* usuários da rede.

Para a montagem da rede utiliza-se os módulos **ovs-ctl** e **ovs-vsctl** do Open vSwitch. Inicialmente inicializa-se o Open vSwitch, depois cria-se a topologia da rede definindo as portas, o controlador e a *bridge* (ponte) entre a rede física e o protocolo OpenFlow. Para isso tem-se os seguintes comandos:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 eth1

ovs-vsctl add-port br0 eth2

Os comandos acima indicam que na *bridge* criada denominada **br0,** criou-se a interfaces da rede OpenFlow nas portas físicas **eth0, eth1** e **eth2** do dispositivo.

Agora é preciso definir o controlador da rede, isto se faz com os seguintes comandos:

ovs-vsctl set-controller br0 tcp :<Endereço IP do controlador em questão>

ovs-vsctl set-fail-mode br0 secure

Os comandos vistos acima significam que foi definido o controlador da *bridge* **br0** pelo protocolo TCP no endereço IP: 192.168.1.10 e para o caso de falha segundo [5] “No OpenFlow, os *switches* pesquisam ativamente um controlador, enquanto o controlador escuta apenas em uma porta especificada (porta 6633 se não estiver configurada diferentemente). A decisão de qual controlador usar é determinada fornecendo o IP do controlador manualmente. Além disso, existe um mecanismo de *fallback* incorporado no Open vSwitch. Se não houver resposta após três tentativas de alcançar o controlador, a *bridge* começará a agir como um *switch* tradicional de MAC *learning* até que a conexão possa ser (re) estabelecida”. Como a ponte é definida como *secure* existe *fallback* e o acesso à rede OpenFlow é protegido.

3.4.2 Definição das *interfaces* da rede

Para estruturar uma rede fixa física para a conexão Open vSwitch nos dispositivos integrantes é necessário nas distribuições Linux acessar e editar o arquivo **/etc/network/interfaces** através do comando **nano** e definir as configurações de rede.

Para o dispositivo Raspberry Pi utilizado como *switch* OpenFlow as configurações foram:

ifconfig br0 <Endereço IP da *Bridge*> netmask <Mascara da Rede> up

3.4.3 Automatização do Open vSwitch

Para automatizar a inicialização do Open vSwitch ao ligar o Raspberry Pi no *boot* do Raspbian, foi realizado a transformação dos comandos correspondentes em um *shell script* com o nome: **openvswitch-bridges.** A sequência de comandos feita foi a seguinte:

sudo nano /etc/init.d/open-vswitch

sudo chmod 755 /etc/init.d/openvswitch-bridges

sudo update-rc.d openvswitch-bridges defaults 21

Após isso, ao reiniciar o Open vSwitch será inicializado sempre no *boot* do sistema Raspbian.

3.4.4 Conexão SSH entre o controlador e o Open vSwitch

Para habilitar uma conexão remota entre o controlador e o *switch* Raspberry Pi montado é necessário antes realizar uma configuração no sistema Raspbian para habilitar o protocolo de comunicação SSH no dispositivo para possibilitar a conexão. Para isso é executada a seguinte sequência:

- no terminal digita-se o comando: **sudo raspi-config;**
- seleciona-se a opção: *interfacing options*;
- seleciona-se a opção SSH;
- por fim a opção *enable* deve ser selecionada;

Agora no terminal do controlador é necessário executar o seguinte comando para estabelecer a conexão SSH em um dado endereço IP do *switch*:

ssh pi@<Endereço IP do Raspberry Pi>

Após isso é necessário entrar com a senha e pronto, a conexão está estabelecida e um terminal no controlador corresponde a um acesso ao dispositivo.

3.4.5 Plataforma Resultante

Após feitas as configurações necessárias, a plataforma resultante com o *switch* OpenFlow Raspberry Pi pode ser vista na Figura15:

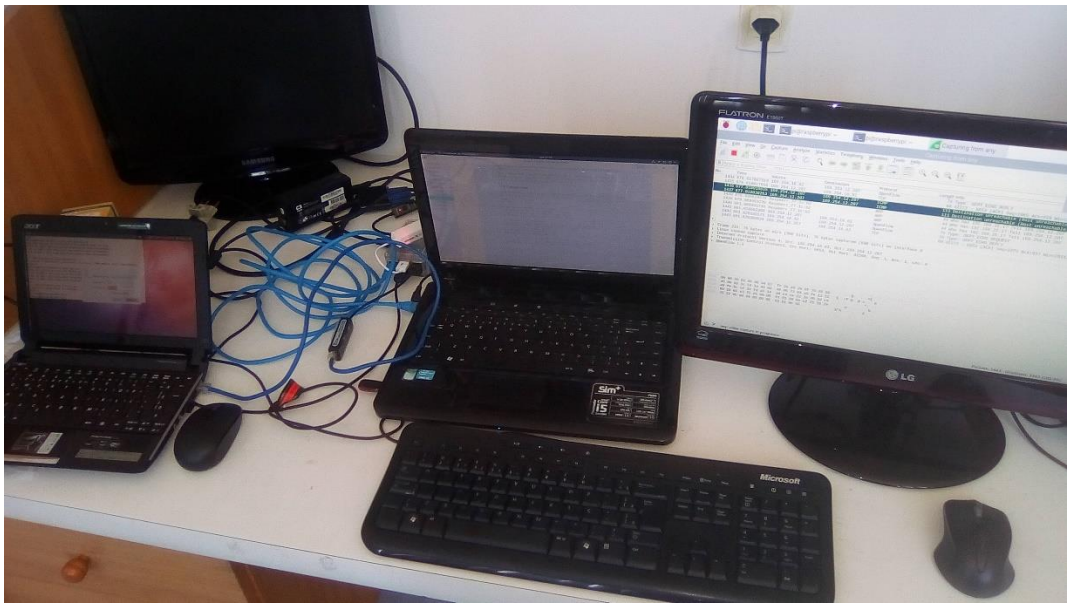

Figura 15. Estrutura de Rede Montada [Autor].

Em relação ao Open vSwitch ao rodar o comando **ovs-vsctl show,** é mostrado a configuração atual do *switch* criado: o nome, as portas, o controlador e a versão. Essa configuração pode ser vista na Figura 16:

```
Arquivo  Editar  Abas  Ajuda
root@raspberrypi:/home/pi# ovs-vsctl show
048b5efc-3317-424c-a205-ab0311153aa5
    Bridge "br0"
        Controller "tcp:192.168.25.238"
            is_connected: true
        Port "eth0"
            Interface "eth0"
        Port "eth2"
            Interface "eth2"
                error: "could not open network device eth2 (No such device)"
        Port "eth1"
            Interface "eth1"
                error: "could not open network device eth1 (No such device)"
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "2.12.0"
root@raspberrypi:/home/pi#
```

Figura 16. Informações da rede Open vSwitch criada [Autor].

3.5 Considerações

A construção do dispositivo *switch* OpenFlow em uma arquitetura Raspberry Pi é um processo de um custo baixo (visto na Seção 3.2) porém com uma série de passos necessários para instalação e configuração para um pleno funcionamento (visto na Seção 3.3).

Neste capítulo foram vistos: uma forma de se montar um *switch* OpenFlow utilizando a arquitetura Raspberry Pi, o sistema Operacional Rasbpian e o *software* Open vSwitch, o processo de instalação e a configuração da plataforma montada.

Capítulo 4

Simulações na plataforma

Neste capítulo serão iniciadas as simulações na plataforma configurada (Cap. 3) primeiramente abordando o funcionamento do *framework* do *software* controlador de SDN utilizado: o RYU. Após isso são especificadas as aplicações de teste que serão utilizadas e qual deverá ser o comportamento esperado dessas durante a execução na rede SDN. Por fim será realizada a análise dos resultados obtidos e uma comparação dessas métricas com outras plataformas que realizam funcionalidades semelhantes às aplicações executadas.

4.1 Framework: Ryu Controller

Sobre o *framework* controlador Ryu, segundo [22] “O RYU é um *framework* baseado em componentes para SDN. O Ryu fornece componentes de *software* com APIs bem definidas que facilitam a criação de novos aplicativos de gerenciamento e controle de rede. Ryu suporta vários protocolos para gerenciar dispositivos de rede, como por exemplo o OpenFlow e é totalmente escrito na linguagem de programação Python”.

A instalação do Ryu pode ser realizada em ambientes Linux ao executar o seguinte comando:

pip install ryu

Ou se preferir instalar diretamente pelo código fonte, executam-se os seguintes comandos:

git clone git://github.com/osrg/ryu.git

cd ryu

pip install

Após isso inicializa-se o executável **ryu-manager,** localizado no caminho: **bin/ryu-manager,** passando como argumento o nome arquivo referente ao código Python baseado no *framework* da aplicação que se deseja executar.

O escopo desse trabalho limitou-se a utilizar os códigos exemplo disponíveis no caminho interno do módulo Ryu baixado do código fonte: **ryu/ryu/app**

4.2 Aplicações teste

Foram executadas aplicações controladoras do *framework* RYU na plataforma construída para prova de conceito e para análise das métricas.

4.2.1 A aplicação Simple_switch_13.py

Essa aplicação tem como função transformar o objeto controlado em um *switch* de rede tradicional de camada de enlace através do protocolo OpenFlow. O código dessa aplicação está no apêndice “A” do trabalho e o funcionamento se dá a partir de, no controlador executar-se o comando seguinte, dado que se esteja no caminho onde o *framework* Ryu foi baixado no terminal:

ryu-manager ryu/ryu/app/simple_switch_13.py --verbose

Após isso, a opção **--verbose** mostra o fluxo de inicialização da aplicação e todos os eventos que se sucedem em relação ao protocolo OpenFlow no controlador. Isso é ilustrado pela Figura 17:

```
root@localhost:/home/eduardo# ryu-manager ryu/ryu/app/simple_switch_13.py --verbose
loading app ryu/ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ryu/ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f99c6f6c710> address:('192.168.25.116', 58876)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f99c6f6c450>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xe948c85,OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id
buffers=0,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 202481593687218 b8:27:eb:77:1c:b2 33:33:00:00:00:16 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 202481593687218 b8:27:eb:77:1c:b2 ff:ff:ff:ff:ff:ff 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 202481593687218 b8:27:eb:77:1c:b2 33:33:00:00:00:16 4294967294
packet in 202481593687218 b8:27:eb:77:1c:b2 33:33:00:00:00:02 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 202481593687218 b8:27:eb:77:1c:b2 33:33:00:00:00:16 4294967294
```

Figura 17. Inicialização de aplicação no Ryu *Controller* [Autor].

Em relação ao protocolo OpenFlow pode-se perceber pela captura dos pacotes trafegados na rede as mensagens do protocolo (Seção 3.1, 3.2 e 3.3) que são trocadas entre o controlador SDN e o *switch* SDN, a comunicação OpenFlow. Isso é ilustrado na Figura 18:

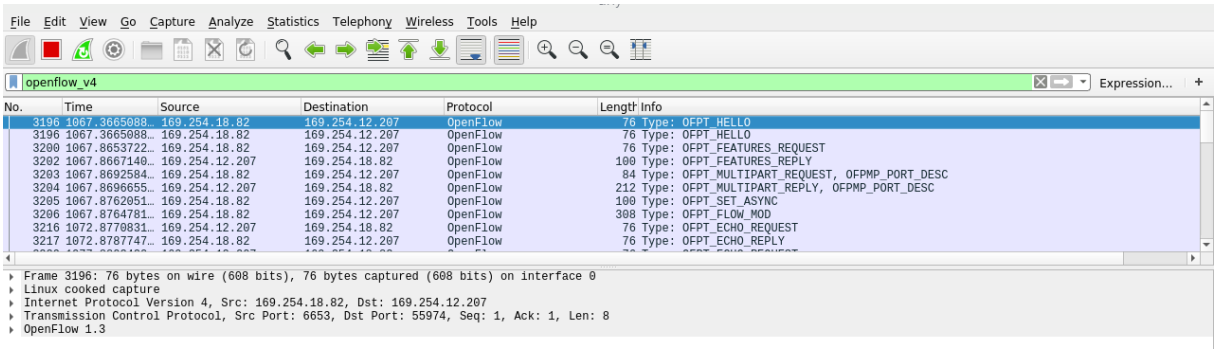

Figura 18. Wireshark: captura de pacotes OpenFlow na rede [Autor].

4.3 Resultados Obtidos

Após a inicialização da aplicação pode-se perceber através do comando **ovs-ofctl dump-flows br0** o estabelecimento da comunicação do controlador com o *switch* OpenFlow em que descreve: qual tabela de fluxo está instalada, o número de pacotes trafegados na comunicação, a prioridade do fluxo, a quantidade de *Bytes* trafegados e o tipo de ação em conjunto com a porta da rede, como ilustra a Figura 19:

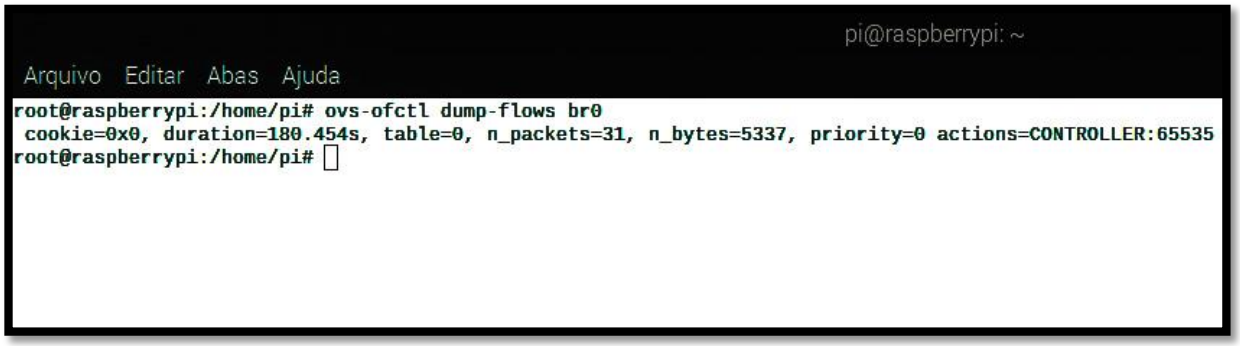

Figura 19. Fluxo inicial entre o controlador e o *switch* [Autor].

Ao se fazer uma requisição de *ping* entre os *hosts* da rede, o *switch* realiza uma rotina de requisitar ao controlador de acordo com a funcionalidade da aplicação, o que deve ser feito em relação ao que será trafegado na rede. Após isso ao executar o

mesmo comando de descrição dos fluxos instalados, obtém-se os novos fluxos definidos pelo controlador na rede como ilustra a Figura 20:

Figura 20. Fluxos instalados pelo controlador da rede [Autor].

Esses fluxos descrevem a funcionalidade da aplicação executada. Nesse caso os fluxos tem como função receber o pacote na porta do campo "*in_port*" e redireciona-lo para a porta de saída "*output"*.

Uma vez ocorridos esses eventos, a comunicação entre os *hosts* será estabelecida. Isso é refletido na resposta do comando de *ping* ilustrada na Figura 21:

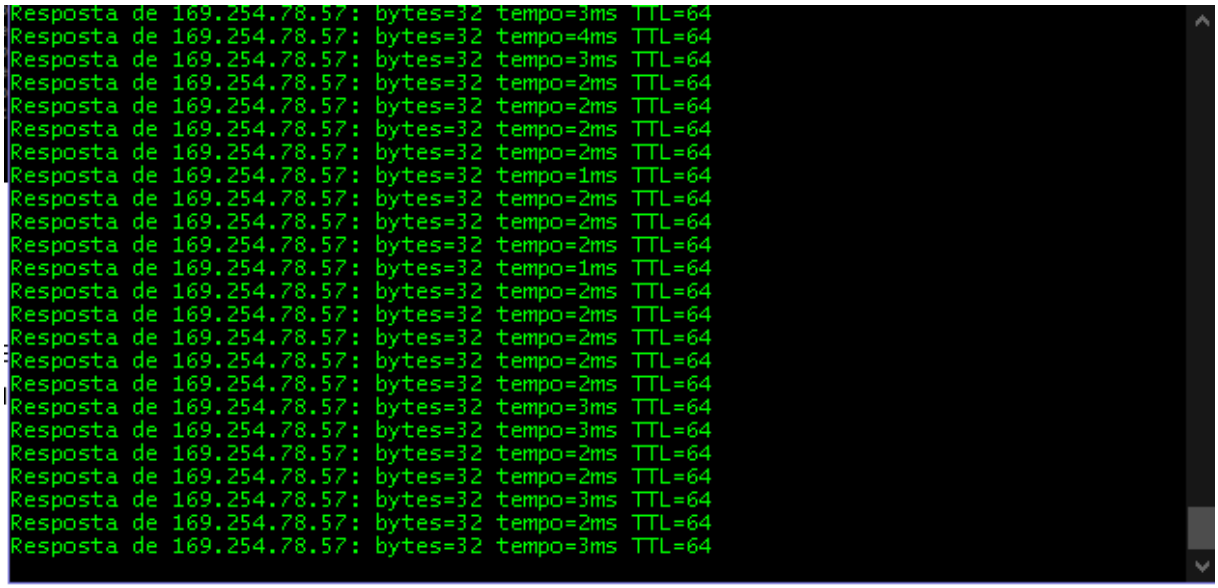

Figura 21. Resposta ao *ping* na rede OpenFlow montada [Autor].

4.4 Considerações

Uma das grandes vantagens de se utilizar redes SDN é a de poder escrever aplicações com uma variedade de *frameworks,* sendo o Ryu um exemplo (seção 4.1), para descrever o comportamento esperado da rede conectada a partir do *switch*

OpenFlow. A partir do fluxo inicial OpenFlow (seção 4.3) o *switch* OpenFlow passa a se comportar como um *switch* de camada 2 do modelo OSI com exceção dos fluxos posteriormente instalados pelo controlador objetivando a funcionalidade escrita na aplicação.

Neste capítulo foi visto uma abordagem sobre o *framework* Ryu para escrita de aplicações SDN, através da execução de uma aplicação exemplo e da demonstração de seus resultados obtidos na rede estruturada

Capítulo 5

Conclusão e Trabalhos Futuros

Visto que o paradigma tradicional de funcionamento das redes de computadores está apresentando uma saturação para os requisitos modernos de controle e de transmissão de dados, o paradigma de redes SDN está em crescente uso e se faz necessário possuir uma plataforma capaz de reproduzir o comportamento de um *switch* SDN com um custo bastante reduzido em comparação as plataformas comerciais, sendo isso o objetivo deste trabalho.

Este trabalho apresentou uma abordagem das redes SDN, seus conceitos básicos e os componentes de uma arquitetura SDN, através da descrição de suas respectivas funcionalidades. Também foi abordado o Protocolo OpenFlow listando-se os tipos de mensagem do protocolo, componentes de um *switch* que suporta esse protocolo e o funcionamento de cada um. Além disso o *software* Open vSwitch utilizado para construção do *switch* virtualizado foi abordado em seus módulos e o processo de execução. Por fim este trabalho apresentou o processo de construção necessário para essa plataforma através de: listagem dos componentes, comandos necessários para instalação e configuração dos componentes da rede estruturada e os resultados obtidos com aplicações de teste na plataforma construída.

5.1 Trabalhos Futuros

O que se espera de contribuições futuras ao trabalho é o aumento do desenvolvimento de aplicações feitas para redes SDN que possam ter o seu ambiente de execução nessa arquitetura ou em outras arquiteturas suportadas. Também pode ser contribuições futuras ao trabalho o aumento de capacidades e de usos para a plataforma construída no campo das SDN como por exemplo: diferentes algoritmos para obter uma melhor forma de controlar e transmitir dados em uma SDN, diferentes protocolos de comunicação entre SDN ou estudos de caso comparando redes tradicionais com redes SDN em diversos aspectos.

Bibliografia

[1] OPEN NETWORKING FOUNDATION. **Software-Defined Networking (SDN) Definition**. Disponível em: https://www.opennetworking.org/sdn-definition/?nab=0 Acesso em: 10 nov. 2019.

[2] Casado, Martin & Koponen, Teemu & Shenker, Scott & Tootoonchian, Amin. (2012). **Fabric: a retrospective on evolving SDN**. DOI: 10.1145/2342441.2342459.

[3] LINUX FOUNDATION. **What Is Open vSwitch**. Disponível em: http://docs.openvswitch.org/en/latest/intro/what-is-ovs/ Acesso em: 21 out. 2019.

[4] McKeown, Nick & Anderson, Tom & Balakrishnan, Hari & Parulkar, Guru & Peterson, Larry & Rexford, Jennifer & Shenker, Scott & Turner, Jonathan. (2008). **OpenFlow: Enabling innovation in campus networks**. Computer Communication Review. 38. 69-74. DOI: 10.1145/1355734.1355746.

[5] SIEBERTZ, Florian. **Masterprojekt - Software Defined Networking**. Disponível em: https://www.h-brs.de/files/20171215_fbinf_mclab_ws14_projektbericht_sdn_siebertz_mk.pdf Acesso em: 8 set. 2019

[6] THE PI. **What is a Raspberry Pi.** Disponível em: https://thepi.io/what-is-a-raspberry-pi/ Acesso em: 2 nov. 2019.

[7] FUJITSU. **Carrier Software DefinedNetworking.** Disponível em: https://www.ofcom.org.uk/__data/assets/pdf_file/0013/32143/sdn_report.pdf Acesso em: 2 nov.2019.

[8] OPEN NETWORKING FOUNDATION. **SDN architecture.** Disponível em: https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf Acesso em: 2 nov. 2019.

[9] OPEN NETWORKING FOUNDATION. **Software-Defined Networking: The New Norm for Networks.** Disponível: https://www.opennetworking.org/wp-

content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf Acesso em: 3 nov. 2019.

[10] OPEN NETWORKING FOUNDATION. **OpenFlow Switch Specification.** Disponível em: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf Acesso em: 15 set. 2019.

[11] THE LINUX INFORMATION PROJECT. **Kernel Definition.** Disponível em: http://www.linfo.org/kernel.html Acesso em: 10 nov. 2019.

[12] PFAFF, Ben Et al. **The Design and Implementation of Open vSwitch.** Disponível em: https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-pfaff.pdf Acesso em: 21 out. 2019.

[13] LINUX FOUNDATION. **Open vSwitch on Linux, FreeBSD and NetBSD**. Disponível em: http://docs.openvswitch.org/en/latest/intro/install/general/ Acesso em: 22 out. 2019.

[14] GIT. **Git Documentation**. Disponível em: https://git-scm.com/docs/git Acesso em: 4 nov. 2019.

[15] M. JONES. **Anatomy of the Linux kernel**. Disponível em: https://developer.ibm.com/articles/l-linux-kernel/ Acesso em: 10 nov. 2019.

[16] **CCNA Certification_Switching**. Disponível em: https://en.wikibooks.org/wiki/CCNA_Certification/Switching Acesso em: 10 nov. 2019.

[17] CISCO. **What Is a Firewall - Cisco.** Disponível em: https://www.cisco.com/c/pt_br/products/security/firewalls/what-is-a-firewall.html Acesso em: 10 nov. 2019.

[18] RED HAT, INC. **What is a Raspberry Pi**. Disponível em: https://opensource.com/resources/raspberry-pi Acesso em: 10 nov. 2019.

[19] RASPBERRY PI FOUNDATION. **Raspberry-Pi-Model-Bplus-Product-Brief**. Disponível em: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf Acesso em: 10 nov. 2019.

[20] OPEN NETWORK FOUNDATION. **OpenFlow Switch Specification 1.4.0.** Disponível em: https://www.opennetworking.org/images/stories/downloads/sdn-

resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. Acesso em: 23 set. 2019.

[21] **Overview OpenFlow**. Disponível em: http://yuba.stanford.edu/cs244wiki/index.php/Overview Acesso em: 15 nov.2019

[22] RYU SDN FRAMEWORK COMMUNITY. **Ryu SDN Framework.** Disponível em: https://osrg.github.io/ryu/ Acesso em: 27 out. 2019.

Apêndice A

Código Fonte: Simple_Switch_13.py

```
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
```

```
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.  The bug has been fixed in OVS v2.1.0.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
```

```
self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

actions = [parser.OFPActionOutput(out_port)]

# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```