INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica,

de Telecomunicações e de Computadores

Aplicações sobre redes definidas por Software baseadas em *OpenFlow*

por

David Alexandre Figueira Pernes

Trabalho Final de Mestrado para Obtenção do Grau de

Mestre em Engenharia de Electrónica e de Telecomunicações

Orientador:

Prof. Doutor Pedro Renato Tavares Pinho

Júri:

Presidente: Prof.ª Doutora Paula Louro Antunes

Vogal-Arguente: Prof. Doutor Fernando M. Ramos

Vogal-Orientador: Prof. Doutor Pedro Renato Tavares Pinho

Novembro 2016

Agradecimentos

Gostaria de agradecer à minha namorada por todo o apoio, paciência e compreensão durante o desenvolvimento desta dissertação e a conclusão do curso de mestrado.

Aos meus pais e irmãos por todo o apoio e conselhos prestados durante todo o percurso académico.

Ao Engenheiro Pedro Pinho pela sua orientação, guia e apoio.

À minha família e amigos pela compreensão da minha ausência em diversas ocasiões.

Aos meus colegas de trabalho pelo seu auxílio e conselhos técnicos.

Resumo

A presente dissertação baseia-se num estudo aprofundado sobre redes definidas por software, mais conhecidas por SDN (Software *Defined Networks*), baseadas no uso do protocolo *OpenFlow*.

Para efetuar este estudo foi analisada a arquitetura de uma rede SDN bem como a comunicação sobre o protocolo OpenFlow fazendo um levantamento de informação relativo a softwares e equipamentos atualmente utilizados.

De maneira a realizar uma prova de conceito sobre as vantagens e potencialidades que uma rede SDN pode oferecer foi implementada uma rede SDN sobre ambiente virtual utilizando o software Floodlight que serve de controlador SDN. Foi também utilizado o software *Mininet* que virtualiza o equipamentos de *switching* com capacidade de comunicação com o controlador via protocolo OpenFlow. Tendo um controlador a comunicar, com sucesso, com diversos equipamentos através do protocolo OpenFlow foi possível aliar conhecimentos de programação java e desenvolver aplicações sobre o controlador.

Foram desenvolvidas 2 aplicações sobre o controlador, sendo que, uma delas teria o objetivo de monitorizar o tráfego de determinadas portas na rede virtualizada e outra teria como objetivo uma a criação de uma GUI (*Graphical User Interface*) para o módulo Firewall disponibilizado pelo controlador. Sendo o Floodlight um software de fonte aberta foi possível efetuar uma contribuição no código fonte do controlador junto da equipa de desenvolvimento da Floodlight no sentido de otimizar o funcionamento do módulo *Firewall*.

Com esta dissertação é possível identificar as possibilidades que uma rede SDN pode oferecer uma vez que este tipo de redes potência a inovação, a flexibilidade e o controlo sobre a rede.

Palavras-chave: Redes definidas por Software, Virtualização, *OpenFlow*, *Floodlight*

Abstract

This thesis aims to address the domain of the SDNs based on the use of the *OpenFlow* protocol in order to gain knowledge in the area and develop applications that take advantage of such networks. In order to develop knowledge in the area a review of the state of the art was made, by reviewing the available literature and the current state-of-the art software-based implementations.

To make a proof of concept about the advantages and the features that a SDN network can offer a SDN network was implemented in a virtual environment using Floodlight software witch servers as a SDN controller and Mininet which virtualizes switching equipment that supports the OpenFlow protocol. Each individual equipment virtualized by Mininet communicates with the server via OpenFlow protocol. With this virtual environment built it was possible do develop software based on java programing language and take advantage of innumerous features that a SDN network can provide.

Two different applications were developed in which one of them consisted in a port monitoring application where it is possible to monitor de current bit rate of a given port and another application that provides a GUI to the controller's firewall module. Since Floodlight is an open source software it was also able to make a contribution in the firewall module in order to optimize its functionality. This contribution was approved by the Floodlight development team.

With this thesis it is possible to verify all the possibilities that a SDN network can offer in terms of innovation, flexibility and control over the network.

Keywords: Software Networks Defined, Virtualization, *OpenFlow*, *Floodlight*

Índice

Agradecimentos iii
Resumo v
Abstract vii
Índice ix
Lista de Figuras xiii
Lista de Tabelas xv
Lista de Acrónimos xvii
Introdução 1
1.1. Enquadramento da Tese 1
1.2. Objetivos da Dissertação 2
1.3. Estrutura da Dissertação 2
1.4. Principais Contribuições 2
Redes SDN 5
2.1 Limitações das redes de telecomunicações 5
2.2 Desenvolvimento do conceito SDN 6
2.3 Arquitetura SDN 7
2.4 Equipamentos SDN 8
2.5 Interfaces de comunicação 9
2.5.1 *Northbound* Interface 10
2.5.2 *Southbound* Interface 10
2.5.3 *Eastbound* / *Westbound* Interfaces 11
2.6 Controladores SDN 11
2.7 *OpenFlow* 13
2.7.1 Equipamento tradicional Vs Equipamento *OpenFlow* 13
2.7.2 *Pipeline* de processamento 14
2.7.3 Tabelas de *Flow* 16
2.7.4 Instruções 17
2.7.5 Ações 18
2.7.6 Tabela de Grupos 19
2.7.7 Tabelas de Medição 20
2.7.8 *OpenFlow Channel* 21
2.8 Limitações do protocolo *OpenFlow* 26
2.9 Aplicações SDN 27

2.10 *Network Functions Virtualization* 28
2.11 Segurança em SDN 28
2.12 Implementação de casos reais 29
2.12.1 Google 30
2.12.2 Facebook 30
2.12.3 Microsoft - Azure 31
Desenvolvimento de Ambiente de Testes 33
3.1 Ambiente de Testes 33
3.2 Seleção de Componente Física 34
3.2.1 Construção de Topologia Virtual 35
3.3 Seleção de Controlador SDN 36
3.3.1 Arquitetura do Controlador *Floodlight* 36
3.3.2 Serviços *Floodlight* 38
3.4 Comunicação *Southbound* 40
3.4.1 Estabelecimento de comunicação *OpenFlow* 41
3.4.2 Ocorrência de *ping* 42
3.4.3 Manutenção do *OpenFlow Channel* 43
3.5 Comunicação *Northbound* 43
3.5.1 Formato JSON 44
3.5.2 Comunicação HTTP 44
3.5.3 Módulos do controlador 49
Implementações Desenvolvidas 51
4.1 *Port Monitor* 51
4.2 *Firewall* GUI 52
4.2.1 *Soft Firewall* vs *Hard Firewall* 53
4.2.2 Interação com módulo *Firewall* 54
4.2.3 Aplicação *Firewall* Gui 55
4.2.4 Comunicação *southbound* das regras 57
4.2.5 Testes de performance 58
4.3 Otimização ao Módulo *Firewall* 58
4.3.1 Formulação do problema 59
4.3.2 Proposta de alteração 59
4.3.3 Aceitação da proposta 63
Conclusões 65
5.1 Trabalho Futuro 66
Referências 67
Anexos 71

A. Diferenças de capacidades entre Versões *OpenFlow* 71
B. Código da alteração ao módulo *Firewall* 71

Lista de Figuras

Figura 1 – Modelo OSI [3] 6

Figura 2 – a) Rede Tradicional b)Centralização do controlo da Rede [3] 7

Figura 3 – Arquitetura SDN [10] 8

Figura 4 – SDN *Interfaces* [15] 10

Figura 5 – Interfaces *westbound* e *eastbound* 11

Figura 6 – Equipamento *switch*ing/routing tradicional [18] 13

Figura 7 – *OpenFlow Pipeline [18]* 15

Figura 8 – Tabela de *flows* extraída a partir do controlador HP VAN SDN 17

Figura 9 – Correspondência de pacotes e execução de instruções numa tabela de *flow [19]* .. 18

Figura 10 – Fluxo de medição das entradas de medição [19] 21

Figura 11 – Componentes principais de um *OpenFlow Switch* [19] 22

Figura 12 - Estabelecimento de ligação *OpenFlow* 25

Figura 13 - Evolução do Protocolo *OpenFlow* a cada Versão [21] 27

Figura 14 – Exposição das principais ameaças a uma rede SDN. [38] 29

Figura 15 – *Wedge switch [9]* 31

Figura 16 – TOR *Switch* “6-Pack” desenvolvido pelo *Facebook*. [9] 31

Figura 17 – ONS 2015: Mark Russinovich 32

Figura 18 – Descrição da Topologia do Ambiente de Testes 33

Figura 19 – *Script* que define uma topologia personalizada. 35

Figura 20 – Ilustração da topologia apresentada. 36

Figura 21 – Arquitetura do Controlador *Floodlight [42]* 37

Figura 22 – Interface Web do controlador *Floodlight*, *Dashboard* 38

Figura 23 – Interface Web do controlador *Floodlight*, *Topology* 39

Figura 24 – Interface Web do controlador *Floodlight*, *Switches* 39

Figura 25 – Topologia Mininet 41

Figura 26 – Captura de Tráfego do estabelecimento de uma comunicação *OpenFlow* 42

Figura 27 - *Packet in* - ARP *Request* 42

Figura 28 – Captura de um pacote *OpenFlow Echo Request* 43

Figura 29 – Captura de um pacote *OpenFlow Echo Reply* 43

Figura 30 – Pedido GET para o endereço http://192.168.1.187:8080/wm/core/controller/*switches*/json 46

Figura 31 – Resposta do servidor ao pedido GET de informação de *switches* 46

Figura 32 – Pedido de Alteração de estado do módulo *Firewall* 46

Figura 33 – Resposta ao pedido de alteração de estado 47

Figura 34 – Pedido POST de uma nova regra no módulo *Firewall* 47

Figura 35 – Resposta do servidor a um pedido POST bem sucedido. 47

Figura 36 - Resposta do servidor a um pedido POST sem sucesso. 48

Figura 37 – Pedido DELETE a um determinado elemento do controlador *Floodlight* 48

Figura 38 – Resposta com sucesso do servidor a um pedido do tipo DELETE bem sucedido. 48

Figura 39 – Resposta do controlador a um pedido DELETE sem sucesso 49

Figura 40 – Módulos *Floodlight [42]* 49

Figura 41 – Aplicação de Monitorização de portos em *switches* 51

Figura 42 – Solução típica de uma rede com uma *Firewall* física. 53

Figura 43 – Solução *OpenFlow* para um *Firewall* virtualizada. 54

Figura 44 - Aplicação *Firewall* GUI 55

Figura 45 – Processamento de uma mensagem ARP numa rede *OpenFlow* 57

Figura 46 – Execução de 1000 *pings* entre 2 *Hosts* sem *Firewall* ativa 58

Figura 47 – Execução de 1000 *pings* entre 2 *Hosts* com *Firewall* ativa 58

Figura 48 – Fluxograma da aceitação de uma regra na *Firewall* do controlador *Floodlight* ... 59

Figura 49 – Novo fluxograma de aceitação de regras na *Firewall Floodlight* 60

Figura 50 – Capacidades das diferentes versões *OpenFlow* [10] 71

Lista de Tabelas

Tabela 1 – Exemplos de equipamentos *OpenFlow* 9

Tabela 2 – Principais características de softwares controladores [10] 12

Tabela 3 – Características de um *Flow* [19] 16

Tabela 4 – Características de um *Group Table* [19] 19

Tabela 5 – Principais componentes de uma *meter entry* [19] 20

Tabela 6 – Principais características de uma banda de medição 21

Tabela 7 – Especificações dos tipos de pedido REST 45

Tabela 8 – Parâmetros presentes nas regras a comparar e valor de bit atribuído nas variáveis *overlap* e *sameField* 61

Tabela 9 – Cenário de sobreposição de regras. 62

Tabela 10 – Sobreposição em múltiplos parâmetros. 62

Tabela 11 – Regras não entram em sobreposição 62

Tabela 12 – Sobreposição de regras mutua. 62

Tabela 13 – Regras Diferentes 62

Lista de Acrónimos

ACL – *Access Control List*

API – *Application Program Interface*

ARP – *Address Resolution Protocol*

CDP – *Cisco Discovery Protocol*

CG-GMS – *Cisco Connected Grid Management System*

CPE – *Customer Premises Equipment*

CPU – *Central Processing Unit*

DDoS – *Distributed Denial Of Service*

DNS – *Domain Name System*

DPID – *Datapath ID*

DSCP – *Differentiated Services Code Point*

EIGRP – *Enhanced Interior Gateway Routing Protocol*

FIB – *Forwarding Information Base*

FPGA – *Field Programmable Gate Array*

GSM – *Global System for Mobile*

GUI – *Graphical User Interface*

HP – *Hewlett-Packard*

HTML – *HyperText Markup Language*

HTTP – *Hypertext Transfer Protocol*

ICMP – *Internet Control Message Protocol*

IDE – *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronics Engineers*

IP – *Internet Protocol*

ITU – *International Telecommunications Union*

JSON – *JavaScript Object Notation*

LAN – *Local Area Network*

LTE – *Long Term Evolution*

MAC – *Medium Access Control*

MEF – *Metro Ethernet Forum*

MPLS – *Multi-Protocol Label Switching*

NETCONF – *Network Configuration Protocol*

NFV – *Network Functions Virtualization*

NIC – *Network Controller Interfaces*

NOS – *Network Operating System*

OCP – *Open Networking Foundation*

OMS – *Optical Management System*

ONF – *Open Networking Foundation*

ONS – *Open Networking Summit*

OSI – *Open Systems InterConnection*

OSPF – *Open Shortest Path First*

OTN – *Optical Transport Network*

OVSDB – *Open Virtual Switch Database Management*

PBB – *Provider Backbone Bridge*

PDH – *Plesiochronous Digital Hierarchy*

POTS – *Plain Old Telephony System*

QoS – *Quality of Service*

RFC – *Request for Comments*

SDH – *Synchronous Digital Hierarchy*

SDN – *Software Defined Network*

SNMP – *Simple Network Management Protocol*

SO – Sistemas Operativos

SSH – *Secure Shell*

TCP – *Transport Control Protocol*

TL1 – *Transaction Language 1*

TLS – *Transport Layer Security*

TNMS – *Transport Network Management System*

TOR – *Top-of-Rack*

TTL – *Time-to-Live*

UDP – *User Datagram Protocol*

URI – *Uniform Resource Identifier*

VLAN – *Virtual Local Area Network*

WDM –*Wavelength Division Multiplex*

xDSL – *Digital Subscriber Line*

XML – *Extensible Markup Language*

WWW – *World Wide Web*

Capítulo 1

Introdução

O mercado das telecomunicações é sem dúvida uma das indústrias que mais se desenvolveu nas últimas décadas. As comunicações entre dispositivos começaram por ter alcances curtos para passarem a atravessar milhares de quilómetros, começaram por ser fixas e passaram a torna-se móveis, começaram por servir dezenas de empresas potenciando o seu desenvolvimento, tornando-se hoje um bem quase tão presente no dia-a-dia da sociedade moderna como a eletricidade ou a água. O leque de serviços tornou-se de tal maneira variado que num passado recente, ano após ano, têm surgido novas ofertas de serviços em vários níveis sendo a voz, o vídeo, a transferência de dados e *cloud* alguns exemplos.

1.1.Enquadramento da Tese

Tecnologias como: xDSL (*Digital Subscriber Line*), POTS (*Plain Old Telephony System*), PDH/SDH (*Plesiochronous/Synchronous Digital Hierarchy*), *ethernet*, GSM (*Global System for Mobile*), WDM (*Wavelength Division Multiplex),* OTN (*Optical Trans*port *Network*) e ainda o LTE (*Long Term Evolution*) constituíram os principais avanços tecnológicos contribuindo ativamente para a introdução de novos paradigmas no mercado. Do ponto de vista das operadoras de telecomunicações, esta constante evolução traduziu-se em elevados custos operacionais no que toca à introdução de novo equipamento compatível com estes novos paradigmas que iriam sendo introduzidos na indústria. Neste âmbito vários fabricantes desenvolveram as mais diversas soluções com base nas tecnologias que seriam apresentadas e normalizadas pelas diversas entidades reguladoras.

Tipicamente a introdução de equipamentos segue uma arquitetura proprietária, com software proprietário e com aplicações proprietárias fidelizando as operadoras de telecomunicações a determinados fabricantes. De maneira a manter certas funcionalidades disponíveis, além de dificultarem o trabalho de manutenção e operação sobre estes equipamentos dado ser necessário, manter sistemas de gestão compatíveis com os mais diversos equipamentos.

De maneira a endereçar estes problemas, surge o paradigma das SDN que possuem como principal objetivo, tornar as redes de telecomunicações um ambiente altamente dinâmico, versátil e convergente abrindo portas à programabilidade da rede de forma centralizada. Uma operadora de telecomunicações introduzindo este paradigma consegue:

- Monitorizar diversos equipamentos de diferentes fabricantes;
- Provisionar em massa diversos equipamentos ao mesmo tempo;
- Desenvolver as suas próprias aplicações;

Neste âmbito o presente documento visa apresentar um estudo sobre os benefícios da introdução deste novo paradigma procurando desenvolver cenários reais ou virtualizados aplicando esta tecnologia.

1.2.Objetivos da Dissertação

A presente dissertação baseia-se no desenvolvimento de conhecimentos aprofundados sobre o conceito de rede SDN, a sua arquitetura, os seus elementos e os seus protocolos de comunicação tendo em vista a criação de aplicações que possam tirar partido das funcionalidades oferecidas por uma rede SDN.

No sentido de alcançar os objetivos propõe-se que seja obtido conhecimento sobre o conceito SDN e sobre os protocolos mais apropriados para a comunicação SDN. Numa segunda fase, tenciona-se que seja implementada uma rede SDN em ambiente virtual, onde sejam disponibilizadas todas as capacidades de uma rede SDN com o intuito de ser possível extrair os mais diversos dados no sentido de fomentar a criação de aplicações externas à rede por forma a tirar partido das mais diversas funcionalidades que uma rede SDN pode oferecer.

1.3.Estrutura da Dissertação

O presente documento encontra-se organizado em 5 capítulos e dois anexos.

Capitulo 1 – No primeiro capítulo do presente documento é realizado o enquadramento da tese no contexto das telecomunicações bem como a exposição dos objetivos que a presente dissertação se propõem a alcançar.

Capitulo 2 – Com o capítulo 2 apresenta-se o estado da arte em torno do conceito das redes SDN, a sua arquitetura, os protocolos utilizados e ainda os diversos elementos que constituem uma rede SDN.

Capitulo 3 – Neste capítulo é apresentado o ambiente de testes desenvolvido para criação de uma prova de conceito. São apresentadas todas as ferramentas utilizadas explicando com elevado detalhe todas as formas de comunicação entre as diferentes plataformas.

Capitulo 4 – Trata-se do capítulo que apresenta todo o trabalho desenvolvido em torno do controlador *Floodlight*. Neste capítulo é possível identificar 2 aplicações desenvolvidas para efeitos de monitorização de tráfego e controlo de tráfego respetivamente. Também neste capítulo é exposta uma falha no controlador *Floodlight* ao nível do módulo de *Firewall* e a contribuição da presente dissertação no sentido de resolver a falha com recurso à apresentação de um novo algoritmo.

Capitulo 5 – Neste capítulo apresenta-se uma síntese de todo o trabalho desenvolvido identificando os problemas ultrapassados e propostas de trabalho futuro em torno da temática apresentada.

1.4.Principais Contribuições

No decorrer do desenvolvimento da dissertação foi proposta a utilização do controlador SDN *Floodlight* para apoio à construção de um ambiente de testes. Após conclusão dessa aplicação constatou-se que o módulo *Firewall* do referido controlador apresentava um modo de aceitação de regras pouco correto e bastante confuso pelo que foi tomada a iniciativa de desenvolver código e apresenta-lo à *Floodlight*.

O código desenvolvido foi submetido para aprovação tendo sido aceite pela equipa de desenvolvimento da *Floodlight*. A formulação do problema e o detalhe da algoritmia apresentada à *Floodlight* encontra-se detalhado no capítulo 4.3 do presente documento enquanto o código fonte desenvolvido que representa a alteração encontra-se no anexo B.

Capítulo 2

Redes SDN

O desenvolvimento do conceito de SDN teve a sua origem na universidade de *Stanford* onde no final do ano 2006, Martin Casado, estudante Ph.D., apresentou um projeto chamado de *Ethane* [1]. O projeto teve como principal objetivo a criação de mecanismos que promovessem uma maior simplicidade à forma como as politicas de policiamento de tráfego eram executadas e aplicadas de modo a tornar a redes mais simples.

2.1 Limitações das redes de telecomunicações

Hoje em dia as redes de telecomunicações são constituídas por um grande leque de equipamentos compatíveis com as mais diversas tecnologias inovadoras que a indústria nos presenteou nas últimas décadas. Estes equipamentos por sua vez são desenvolvidos por fabricantes diferentes como por exemplo a Nokia, a Coriant e a Cisco Systems. Os equipamentos desenvolvidos normalmente possuem software proprietário compatível com as normas aprovadas por entidades como a ITU (*International Telecommunications Union*), IEEE (*Institute of Electrical and Electronics Engineers*) e o MEF (*Metro Ethernet Forum*) entre outras.

No que respeita à operação e manutenção sobre equipamentos tradicionalmente são utilizados protocolos de comunicação como o *telnet* e o SSH (*Secure SHell*) no entanto apenas de forma iterativa, isto é, apenas se configura um equipamento de cada vez. Após ser possível o acesso à configuração de um equipamento, um operador tem ao seu dispor um software que normalmente difere de fabricante para fabricante. Regra geral, os equipamentos utilizam SO (Sistemas Operativos) baseados em *Linux* no entanto os comandos e as configurações diferem de fabricante para fabricante.

Algumas aplicações como *ProVision Network Management System*, *Alcatel-Lucent 1350* OMS (*Optical Management System*), CG-NMS (*Cisco Connected Grid Management System*) e TNMS (*Trans*port *Network Managment System*) são exemplos de aplicações de alto nível proprietários que tornam tarefas de operação, manutenção e monitorização de rede mais simples.

De maneira a tornar a operação e manutenção algo mais simples e agnóstica face a fabricantes, são utilizados protocolos como o SNMP (*Simple Network Management Protocol*), CDP (*Cisco Discovery Protocol*), NETCONF (*Network Configuration Protocol*) e TL1 (*Transaction Language 1*). Estes protocolos são ainda hoje bastante utilizados e diversas aplicações de alto nível baseiam-se neles de modo a efetuarem funções muito importantes na gestão da rede como a monitorização de alarmística da rede, distribuição de listas de acesso, configuração massiva em diversos equipamentos, etc. Embora sejam protocolos de utilidade notável por vezes os mesmos podem não ser utilizados por incompatibilidade do equipamento ou por determinado fabricante impor a utilização de protocolos proprietários como o TL1, tipicamente mais presente em equipamentos do fabricante Nokia, enquanto o CDP exclusivo a equipamentos do fabricante Cisco.

Atualmente, a heterogeneidade de fabricantes e tecnologia adiciona complexidade à rede tornando tarefas simples de operação e manutenção algo penosas e com custos bastante elevados, dado ser necessário despender mais tempo para as executar. Neste sentido, existe uma forte necessidade de tornar procedimentos de operação e manutenção mais autónomos e com a menor intervenção humana possível, de maneira a que as redes tradicionais, se tornem altamente flexíveis, mutáveis e adaptáveis face às exigências do mercado.

Com efeito, as redes tradicionais possuem:

- Elevada heterogeneidade de equipamentos /tecnologia;
- Configuração iterativa, pouco flexível e suscetível a erros;
- Diversos sistemas de gestão de alto nível para monitorização de rede;
- Elevada incompatibilidade entre sistemas de monitorização / aplicações de alto nível;
- Configurações de elevada complexidade;
- Falta de um mecanismo de configuração comum a todos os equipamento.

2.2 Desenvolvimento do conceito SDN

Desde os anos 70 [37] que os protocolos de comunicação têm seguido de perto o modelo OSI (O*pen Systems Interconnection*) como ilustrado na figura 1. Neste modelo, todas as funções de uma rede são descritas segundo um modelo de 7 camadas, sendo que, cada função que ocupe uma camada superior terá obrigatoriamente de utilizar as inferiores. Cada camada é acompanhada de alguns exemplos de protocolos.

Figura 1 – Modelo OSI [3]

Segundo [4], analisando este modelo identifica-se que as camadas 1 a 3 estão sempre presentes em toda a rede, enquanto as camadas 4 a 7 remetem sempre para serviços do tipo extremo a extremo. Sobre o ponto de vista do desenvolvimento de um mecanismo de controlo centralizado das políticas da rede, verifica-se que o mecanismo terá de atuar fundamentalmente sobre as camadas 1 a 3 de maneira a garantir total

controlo do fluxo de tráfego. Através do controlo centralizado das camadas 1 a 3, torna-se possível otimizar procedimentos e melhorar as diversas funções de operação e manutenção, como por exemplo a implementação de soluções na rede de forma transversal onde é possível configurar todos os equipamentos de uma rede através desse mesmo controlo centralizado.

Como já mencionado o projeto-piloto de desenvolvimento de uma SDN teve a sua origem em 2006 durante um doutoramento realizado por Martin Casado, tendo desenvolvido um sistema denominado *Ethane*. O projeto *Ethane* teve como principais objetivos a centralização do controlo de uma rede facultando simultaneamente uma maior segurança e flexibilidade à rede. Com o *Ethane* os utilizadores poderiam implementar policiamentos de tráfego, altamente detalhados, diretamente na rede de uma forma global e genérica em vez de se ter de efetuar uma configuração específica e detalhada para cada equipamento através da imposição de regras simples denominadas por *flow*.

2.3 Arquitetura SDN

De maneira a que esta visão de rede possa ser implementada é necessário que as redes sejam trabalhadas de outra forma, isto é, tradicionalmente os equipamentos possuem a definição dos serviços, o controlo e mantêm a noção da rede, o que se traduz em maior carga em termos de processamento para os equipamentos traduzindo-se no encarecimento do mesmo, no aumento de latência e no aumento da complexidade da arquitetura dos equipamentos. A figura 2 a) ilustra as principais responsabilidades dos *switch* numa rede tradicional enquanto a figura 2 b) ilustra a remotização de algumas funções do *switch*.

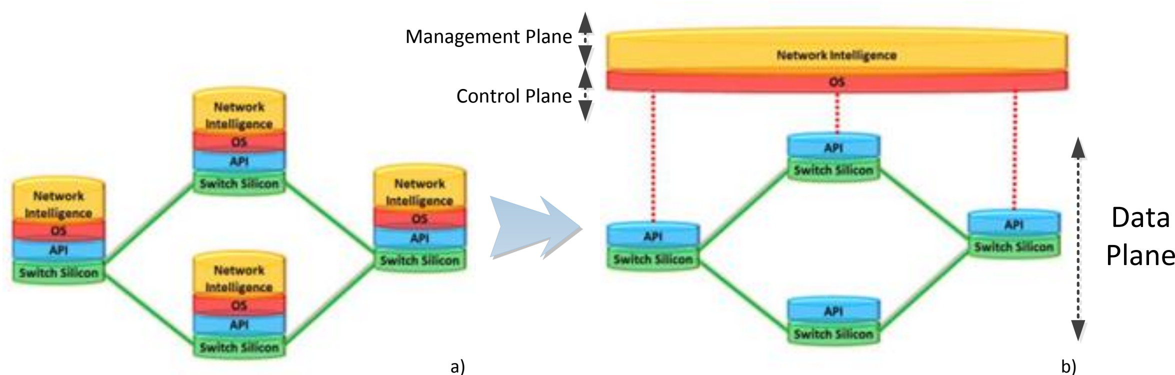

Figura 2 – a) Rede Tradicional b)Centralização do controlo da Rede [3]

A arquitetura SDN é definida pelos planos de gestão, controlo e dados

O plano de dados é constituído pela infraestrutura de rede, isto é, equipamentos, cabelagens e todos os restantes componentes de hardware que permitem a comutação de pacotes.

O plano de controlo é constituído por um ou mais servidores portadores de um NOS (*Network Operating System*) vulgarmente chamado de controlador. Para além dos controladores, sobre os servidores podem ainda atuar hipervisores de rede que trazem

novas funcionalidades à rede através da implementação de protocolos sobre os diversos equipamentos da rede, como por exemplo a virtualização de redes sobre a rede física através de VLAN *(Virtual Local Area Network)*, definição de listas de acesso dinâmicas entre outras funcionalidades. O controlador permite que os administradores de rede possam programar sobre a rede permitindo o desenvolvimento de diversas aplicações como por exemplo:

- Criação de esquemas de proteção sofisticados;
- Criação de *Firewall* localizadas em qualquer ponto da rede;
- Criação de ferramentas de monitorização de tráfego;
- Balanceadores de tráfego

Este plano deve ser encarado como o plano nuclear de uma SDN dado que estabelece interfaces de comunicação entre o plano de gestão e o plano de dados. A comunicação do plano de controlo com o plano de dados é assegurada por protocolos que aplicam regras e decidem sobre o que é permitido circular na rede ou não.

Relativamente ao plano de gestão, através das informações que são disponibilizadas pelo plano de controlo, é possível que sejam desenvolvidas aplicações que extraiam informações sobre a rede em tempo real. É no plano de gestão que se abrem portas à programabilidade e à inovação na rede, permitindo aos operadores o desenvolvimento das mais diversas aplicações.

Na figura 3 são descritas interfaces denominadas de *northbound* e *southbound* interfaces. É através destas interfaces que a comunicação é assegurada entre os diferentes planos.

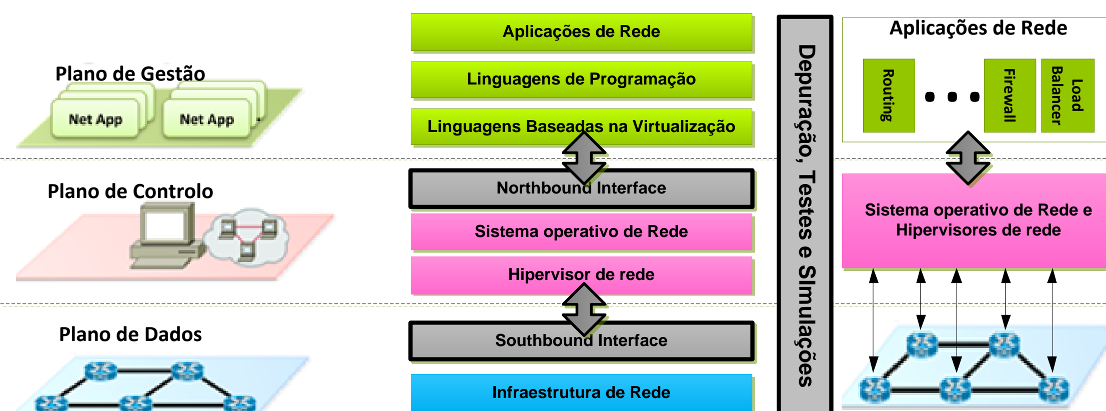

Figura 3 – Arquitetura SDN [10]

2.4 Equipamentos SDN

O plano de dados é fundamentalmente constituído por diversos equipamentos que formam uma rede de telecomunicações. Um equipamento SDN por norma tem de possuir uma interface de comunicação com um controlador SDN, isto é, tem de suportar uma comunicação via *southbound* interface.

Atualmente, no mercado, existem diversos equipamentos que oferecem suporte a protocolos como OVSDB (*Open Vswitch Database Management*) e *OpenFlow,* no sentido de permitir a administradores a implementação de uma rede SDN. São já

diversos fabricantes que possuem a sua presença no mercado ao disponibilizar o mais variado leque de equipamentos, desde CPE (*Customer Permises Equipment*), a *service routers* que permitam todas as funcionalidades SDN.

Na tabela 1 verificamos alguns produtos de diversos fabricantes, o que revela uma certa tendência do mercado a utilizar SDN como base para o futuro das redes. Outro aspeto verificado reside na aceitação global por parte dos fabricantes em implementar o *OpenFlow* como protocolo de comunicação na *southbound* interface.

Fabricante	**Equipamento**	**Versão *OpenFlow***	**Descrição**
Nuage Networks	*7850 Services Gateway*	1.3	*Ultra-High density 96x10Gb Gateway*
Pica8	*P-5401*	1.4	*WhiteBox Switch*
Hewlett-Packard	*8200zl and 5400zl*	1.0 e 1.3	*DataCenter Switch*
NoviFlow	*NoviSwitch Series*	1.3 e 1.4	*High performance OpenFlow Switch*
Brocade	*ICX 7450 Switch*	1.3	*Top of Rack Data Center Switch*
Juniper	*EX Switches Series*	1.0 e 1.3	*Wide Variety of OpenFlow enable Switches*

Tabela 1 – Exemplos de equipamentos *OpenFlow*

Tipicamente os equipamentos de encaminhamento e comutação possuem SO proprietário como acontece em equipamentos de diversos fabricantes como a Cisco Systems, Juniper, Nokia entre outros. As SDN permitiram a introdução de *whitebox switches* no mercado por parte de fabricantes como *Pica8*. O termo *whitebox switches* ou *bare-metal switches* descreve equipamento de encaminhamento e comutação de pacotes que não possui SO a correr nativamente, trazendo a possibilidade de um administrador de redes desenvolver, aplicar ou adquirir um SO ao seu gosto. [47]

Outra forma de obter equipamentos que suportem comunicação via *OpenFlow* pode passar pela instalação de software, ou pela atualização de *firmwares* em equipamentos de redes tradicionais. Em alguns casos de estudo como [12] e [13] foi demonstrada a possibilidade de transformar equipamentos de redes tradicionais em equipamentos que suportem *OpenFlow*.

Alternativamente também é possível virtualizar estes equipamentos sobre máquinas de elevada capacidade de processamento de maneira a poder criar ambientes de simulação e testes. O *Mininet* [14], por exemplo, é uma aplicação que permite a criação de uma rede Virtual de forma instantânea o que permite a criação de cenários de simulação e desenvolvimento de redes SDN.

2.5 Interfaces de comunicação

Numa arquitetura SDN a rede centraliza grande parte das suas funções nos seus controladores, como tal são definidas interfaces de comunicação entre as diversas camadas, sendo que, de forma figurativa, a norte do controlador, encontram-se as aplicações, a sul do controlador encontram-se os equipamentos, a este e a oeste encontram-se outros controladores SDN. Segundo ilustrado na figura 4 é possível verificar as interfaces descritas.

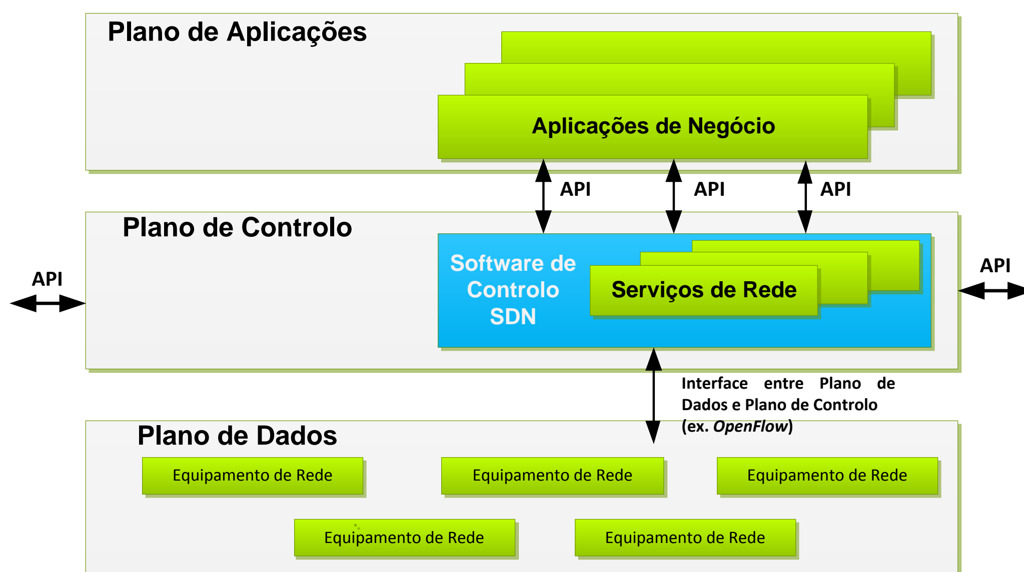

Figura 4 – SDN *Interfaces* [15]

2.5.1 *Northbound* Interface

É através desta interface que a rede aumenta a sua flexibilidade sendo possível programar toda a rede através de um canal único, próprio e seguro.

As *northbound* interfaces compreendem a ligação entre aplicações e controladores SDN. Esta interface normalmente é definida por uma API (*Aplication Program Interface*), através da qual programadores podem definir o seu código de maneira a extrair informações dos equipamentos, definir novas regras de encaminhamento, recolher dados estatísticos, tabelas de endereçamento, entre outras informações.

Dependendo do controlador utilizado, a forma como é efetuada a comunicação entre aplicações e controladores varia. Normalmente a comunicação é efetuada através de linguagens bastante populares como HTTP (*Hypertext Transfer Protocol*), JSON (*JavaScript Object Notation*) e XML (*Extensible Markup Language*). No controlador da *Hewlett-Packard* (HP) SDN VAN a comunicação com aplicações é efetuada através de http GET, POST e DELETE que seguem a estrutura da API facultada pelo fabricante HP [16].

2.5.2 *Southbound* Interface

A *southbound* interface pode basear-se em vários protocolos tais como: *OpenFlow*, OVSDB, ForCES [10] e ainda outros como SNMP, NETCONF, BGP. Suportando vários protocolos, é possível manter a compatibilidade do controlador com um maior número de equipamentos. A maioria dos controladores apenas suporta *OpenFlow* sendo este o protocolo, testado comercialmente e bastante apoiado pela ONF (*Open Networking Foundation*) [17]. De maneira a que a comunicação seja possível ambos os agentes envolvidos na interface (controlador e equipamento) devem suportar uma versão *OpenFlow* comum. No capítulo 2.7 será abordada com maior detalhe a comunicação via protocolo *OpenFlow*.

2.5.3 *Eastbound / Westbound* Interfaces

Tanto a *eastbound* como a *westbound* interface, compreendem a comunicação entre diferentes controladores. Estas interfaces são necessárias em cenários onde uma rede é gerida por múltiplos controladores, isto é, um sistema distribuído, conforme ilustrado na figura 5.

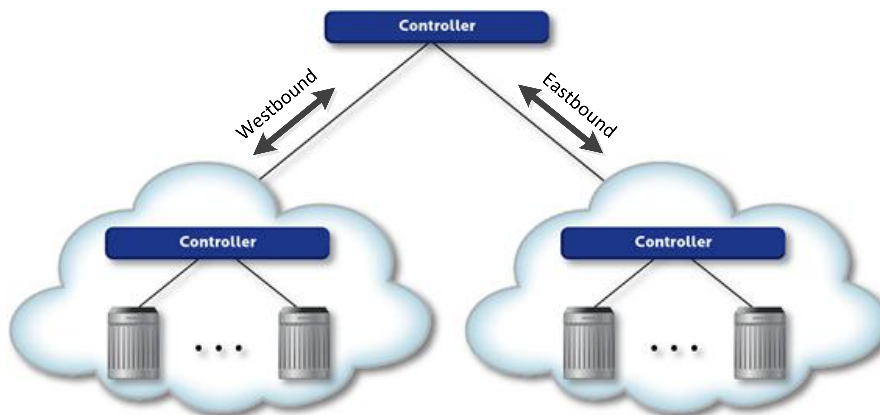

Figura 5 – Interfaces *westbound* e *eastbound*

A principal utilidade destas interfaces reside principalmente no facto de permitir a possibilidade de ativar mecanismos de replicação de dados que permitam o aumento da robustez do sistema SDN em caso de falha do controlador. Outras funções destas interfaces baseiam-se na importação/ exportação de dados, notificações, monitorização e implementação de algoritmos de consistência entre controladores. À semelhança da *northbound interface* a *eastbound* e a *westbound* interface também possui a sua própria API. [10]

Após analisar as funcionalidades de todas as interfaces e verificar as formas como estão definidas é possível concluir que, o meio de comunicação das interfaces depende, fundamentalmente, do controlador escolhido e da sua compatibilidade de comunicação com os equipamentos que constituem o plano de dados.

2.6 Controladores SDN

Um controlador SDN é o centro de toda a operação de uma rede SDN. É através deste elemento que toda a rede centraliza a sua operação e informação tornando o controlador o elemento mais crítico de uma rede SDN pelo que a sua implementação deve ser bastante cuidada no sentido de permitir uma grande disponibilidade de todos os dados que o controlador recolhe da rede e aos recursos que o controlador faculta às aplicações

Um controlador consiste num software que não tem de seguir nenhuma norma, no entanto, um controlador deve possuir a capacidade de estabelecer a comunicação com as camadas de infraestrutura e com a camada de aplicação via *northbound interface* e *southbound interface*, respetivamente.

Regra geral um controlador SDN é um software instalado sobre um servidor de elevada capacidade de processamento que disponibiliza diversas interfaces de comunicação com a camada da infraestrutura e com a camada de aplicações.

A tabela 2 ilustra alguns dos controladores mais difundidos bem como algumas das suas principais características:

Software	**OpenDaylight**	**Floodlight**	**HP VAN SDN**	**Onix**	**Beacon**
Serviços de rede	Topologia/Estatisticas/ Gestão de *Switches*, Localização de *Host*, ShorTest Path Forwarding	Topologia/Estatisticas/ Gestão de Switches, Routing, Forwarding, Firewall, Static Flow Pusher	Relatório de audição, Alertas, Topologia, Descoberta	Discovery, Multi-consistency Storage, Read State, Register for updates	Topologia, Gestão de equipamentos routing
East/WestBound APIs	N/A	N/A	Sync API	Distribution I/O module	N/A
Plugins	OpenStack Neutron	N/A	OpenStack	N/A	N/A
Interfaces de Gestão	GUI/CLI, REST API	Web GUI	REST API Shell / GUI Shell	N/A	Web
***Northbound* APIs**	REST, REST-CONF, *JAVA* API	REST APIs	REST API Shell / GUI Shell	Onix API	API (baseada em eventos *OpenFlow*)
***Southbound* API**	*OpenFlow*, OVSDB, SNMP, PCEP,BGP, NETCONF	OpenFlow	*OpenFlow* L3 Agent, L2 Agent	*OpenFlow*, OVSDB	*OpenFlow*

Tabela 2 – Principais características de softwares controladores [10]

Existem diversos fabricantes de software controlador como é o caso da Juniper, Cisco, Big *Switch*, HP, IBM e VMWare.

Tendo em conta a quantidade e a variedade de controladores existentes a escolha de um controlador a utilizar numa rede deve ser alvo de uma cuidada análise sobre os seguintes aspetos [26]:

- Programabilidade, para garantir uma maior flexibilidade na rede;
- Eficácia, medida através da performance, segurança, escalabilidade e confiabilidade que o controlador oferece;
- Funcionalidades, tais como policiamento de tráfego, balanceamento de carga, monitorização de tráfego, introdução de novos serviços, etc.

2.7 *OpenFlow*

O protocolo *OpenFlow* é a primeira norma aceite pela ONF como interface entre o plano de controlo e o plano de dados definido na arquitetura SDN. Este protocolo permite o acesso e controlo da configuração de *switches* e *routers* de forma centralizada, permitindo a manipulação da forma como diferentes pacotes são encaminhados em *switches* e *routers*.

2.7.1 Equipamento tradicional Vs Equipamento *OpenFlow*

O encaminhamento e comutação de pacotes em equipamentos de redes tradicionais baseia-se na leitura de tabelas, como a tabela MAC (*Medium Access Control*), tabela de encaminhamento IP (I*nternet Protocol*) e listas de acesso. Tradicionalmente, estas tabelas são populadas manualmente ou dinamicamente através de protocolos como o ARP (*Address Resolution Protocol*), OSPF (*Open Shortest Path First*), EIGRP (*Enhanced Interior Gateway Routing Protocol*), entre outros protocolos.

Na figura 6 verificamos a representação gráfica de algumas tabelas num *switch*/*router* onde são identificadas as tabelas que são populadas dinamicamente e as tabelas que são populadas manualmente. Regra geral, estas tabelas variam pouco após a rede ter sido implementada.

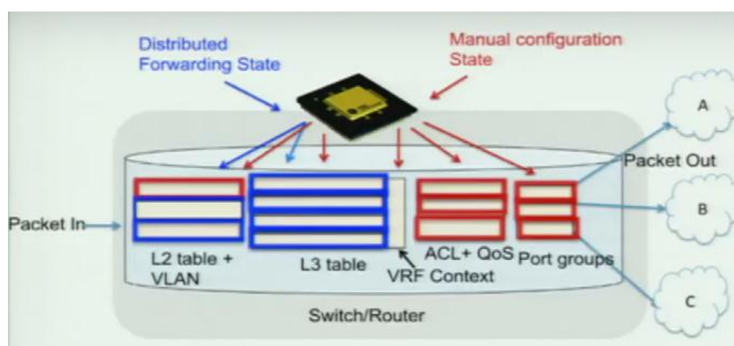

Figura 6 – Equipamento *switch*ing/routing tradicional [18]

No que toca a tabelas populadas manualmente, muitas vezes implicam que um administrador tenha de aceder remotamente a cada equipamento e proceder à sua configuração. Este modo de trabalho torna-se exaustivo e pouco produtivo. O protocolo *OpenFlow* vem precisamente endereçar este problema.

Numa rede SDN a responsabilidade da manutenção destas tabelas passa a estar centralizada exclusivamente no controlador SDN. Todos os conceitos inerentes às tabelas descritas são aplicados numa rede SDN, no entanto, apenas o controlador SDN efetuará o processamento das mesmas libertando os equipamentos dessa responsabilidade. O controlador SDN manterá todas as tabelas atualizadas em virtude das informações que são passadas através dos equipamentos da rede e através do uso de algoritmos como por exemplo o Dijkstra [46]. Um administrador de uma rede SDN poderá ainda definir as suas politicas de tráfego diretamente a partir do controlador sendo essa informação disseminada exclusivamente para os equipamentos em que tais configurações façam sentido.

Um equipamento SDN concentrará todas as suas decisões de encaminhamento com base numa única tabela denominada de tabela de *flow*s. Cada entrada nesta tabela é denominada de *flow*. Um *flow* descreverá qual a ação que um equipamento deverá dar a um determinado pacote. Se por algum motivo um equipamento não conseguir apurar o destino que deverá dar a um pacote com base na sua tabela de *flow*s cabe ao equipamento encaminhar esse pacote para o controlador SDN de maneira ao controlador partilhar a informação necessária de encaminhamento do pacote.

Através do protocolo *OpenFlow*, todos os nós da rede conseguem efetuar atualizações aos seus *flows* (entradas de tabelas) através da troca de mensagens com o controlador

Segundo [18], um *switch OpenFlow* diferencia-se de um *switch* tradicional pela sua capacidade de comunicação com um controlador SDN através do protocolo *OpenFlow*.

Nos seguintes subcapítulos serão abordados com maior detalhe a comunicação entre um controlador e um equipamento de rede, bem como, o processamento dos pacotes de tráfego por parte de equipamentos de uma rede SDN.

2.7.2 *Pipeline* de processamento

O processamento de pacotes em equipamentos *OpenFlow* pode ser de dois tipos: *OpenFlow-Only* ou *OpenFlow Hybrid*. Os primeiros implicam que todos os pacotes devem ser processados pelas tabelas de *flow*, enquanto os segundos possuem duas *pipelines* de processamento. Estes processam pacotes através de tabelas de *flow* mas também através de tabelas de encaminhamento *ethernet* tradicionais com base em tabelas de *routing*, *forwarding*, ACLs etc.

Os pacotes ao serem recebidos procuram correspondências nas tabelas de *flow*. Caso exista correspondência entre os pacotes e as entradas de *flow* os pacotes prosseguem na *pipeline* onde serão aplicadas as ações descritas pelo *flow*. De seguida um pacote poderá ser enviado para uma nova tabela de *flow*, com *table ID* diferente, ou poderão ser executadas as ações de grupo que estejam associadas ao *flow*. Caso o *flow* possua alguma ação de *output* o pacote passa para o *egress* do *switch*.

Ao nível do *egress* um *switch OpenFlow* pode ainda suportar tabelas de *flow* especificas de *egress*. O processamento sobre estas tabelas é bastante semelhante ao das tabelas de *flow* de *ingress*.

A figura 7 descreve o todo o *pipelining* de processamento aplicado ao tráfego processado por *switches OpenFlow*.

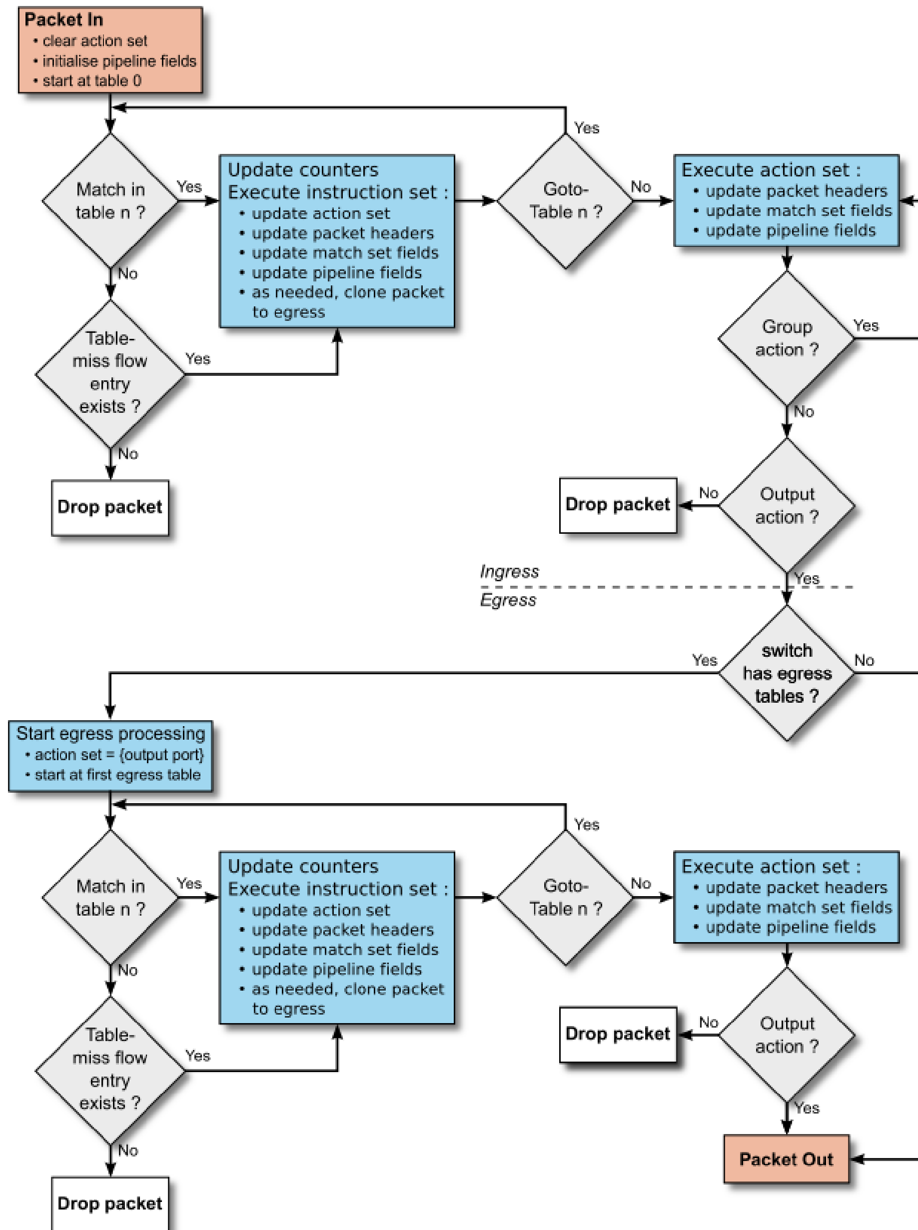

Figura 7 – *OpenFlow Pipeline [18]*

A consistência da informação contida nas tabelas é um ponto critico no correto funcionamento do processamento *OpenFlow*. Para que a consistência seja mantida existem diversas regras que têm de ser obedecidas ao nível das tabelas, das entradas de *flow*, das mensagens trocadas entre *switch* e controlador. Por exemplo, um pacote recebido pelo *switch* não pode ser alterado a menos que explicitamente indicado, tal como na ocorrência de um *table-miss* (omissão de entrada de *flow* correspondente para a um pacote) o *switch* solicita uma entrada de *flow* ao controlador, este por sua vez terá de indicar ao *switch* uma entrada de *flow* apropriada para o pacote que originou o *table-miss*.

2.7.3 Tabelas de *Flow*

Inicialmente o protocolo OpenFlow na sua versão 1.0 apenas suportava uma única tabela de flow enquanto atualmente, na versão 1.5 o protocolo *OpenFlow* pode ser configurado sobre várias tabelas, entre as quais as, tabelas de *flow*. Esta tabela consiste num conjunto de regras que possuem diversos parâmetros, contadores e outras indicações. Cada entrada desta tabela é denominada um *flow*. Um pacote ao ser recebido num *switch* é submetido a uma tentativa de correspondência com um *flow* para que seja despoletada alguma ação.

A tabela 3 indica os parâmetros que constituem um *flow*

Match Field	*Priority*	*Counters*	*Instructions*	*Timeouts*	*Cookie*	*Flags*

Tabela 3 – Características de um *Flow* [19]

Cada *flow* contem os seguintes campos:

- ***Match Field*** – conjunto de parâmetros a serem comparados com um pacote recebido. Através da correspondência destes parâmetros com os dados presentes nos pacotes recebidos é que se determina se a ação despoletada por um *flow* é executada ou não executada. Neste conjunto de parâmetros podem ser definidos endereços MAC, endereços IP, portos TCP (*T*ransport Control Protocol) / UDP (*User Datagram Protocol*), porto de origem, *Ethertype*.
- ***Priority*** – Precedência do *flow* na tabela;
- ***Counters*** – Indica o número de pacotes que um *flow* já identificou;
- ***Instructions*** – Indica ações que alteram o processamento de um determinado pacote, ver capítulo 2.7.4;
- ***Timeout*** – Tempo máximo permitido para a um *flow* permanecer ativo;
- ***Cookie*** – Campo preenchido pelo controlador. Pode ser utilizado para filtrar entradas de *flow*. Este campo não é utilizado no processamento de pacotes;
- ***Flags*** – Alteram a forma como um *flow* é gerido.

Na figura 8 verifica-se um exemplo de uma tabela de *flows* retirada da implementação de um controlador com interface gráfica desenvolvida pela HP. A tabela ilustrada possui 3 *flows* diferentes onde o 1º *flow* irá corresponder todos os pacotes IPv4 que utilizem o protocolo ICMP (*Internet Control Message Protocol*) não executando nenhuma ação servindo apenas para fins estatiscos. Todos os pacotes que possuem campos iguais aos campos presentes na coluna *Match Fields* de um *flow* provocam o incremento de *pacotes* e *bytes*. O 2º *flow* efetua a tentativa de correspondência a todo o tráfego IPv4 que possua o protocolo UDP com porto origem 68 e porto destino 67. Em caso de correspondência o *switch* deverá enviar o conteúdo desse pacote para o controlador e ainda deverá propagar o pacote como se o *switch* tivesse um comportamento normal. Já o 3º *flow* efetua as mesmas ações que o 2º *flow* no entanto possui uma diferença ao nível dos portos UDP origem e destino. Tanto o 2º *flow* como o 3º *flow* possuem número de prioridade mais alto pelo que devem ser verificados por último.

Table ID	Priority	Packets	Bytes	Match Fields	Actions
▸ 0	10000	25	2450	eth_type: ipv4 ip_proto: icmp	
▸ 0	31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 68 udp_dst: 67	apply_actions: output: CONTROLLER output: NORMAL
▸ 0	31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 67 udp_dst: 68	apply_actions: output: CONTROLLER output: NORMAL

Figura 8 – Tabela de *flows* extraída a partir do controlador HP VAN SDN

A norma prevê ainda a existência de uma ou mais tabelas de *flows*. Estas tabelas são distinguidas pela existência de um identificador denominado de *table* ID.

2.7.3.1 Correspondência de Pacotes

Ao dar entrada num *switch OpenFlow,* um pacote, é analisado e submetido a uma tentativa de correspondência nas tabelas de *flow* presentes num *switch*. A procura de uma correspondência é efetuada por comparação de cada pacote com cada entrada de *flow* (*flow entry*) presente na tabela de *flows* obedecendo às prioridades presentes na tabela. Um pacote corresponde com uma entrada de *flow* quando os parâmetros de um dado pacote possuem os mesmos dados presentes no campo match field de um *flow*. Na eventualidade de existirem 2 entradas de *flow* com *match fields* que correspondam ao pacote, será correspondida a entrada de *flow* que possui maior prioridade.

Numa tabela não devem existir entradas de *flow* com *match fields* e prioridades iguais.

2.7.3.2 Falhas de Correspondência

Por defeito todas as tabelas de *flows* têm de suportar falhas de correspondência. Segundo a norma [19], uma falha de correspondência denomina-se de *table-miss*. Uma entrada de *table-miss* presente numa tabela de *flows* deverá ser a entrada menos prioritária, com *match fields* a aceitar qualquer valor e ainda deverá enviar uma mensagem ao controlador no sentido de solicitar uma atualização às suas tabelas de *flows* de maneira a apurar qual será o destino de um pacote que tenha caído numa entrada de *table-miss*.

Todos os *switch OpenFlow* que não possuam uma entrada de *table-miss* devem descartar o pacote recebido dado que não ocorreu correspondência com qualquer uma das entradas de *flow*.

2.7.4 Instruções

Um *switch OpenFlow,* após efetuar a correspondência entre um pacote e uma entrada de *flow,* deverá executar o conjunto de listas de ações presentes na entrada de *flow*. Uma instrução pode alterar a forma como um pacote é tratado dentro da tabela de *flow*. Segundo a tabela 3, os *flows* são caracterizados também pelas suas instruções.

Uma instrução pode ser do seguinte tipo:

- ***Apply-Actions*** – Aplica um conjunto de ações imediatamente. Esta instrução pode ser utilizada para alteração de pacotes entre duas tabelas ou executar ações múltiplas do mesmo tipo. As ações mencionadas estão especificadas na secção 3.4.4;
- ***Clear Actions*** – *Remove* todas as ações da lista de ações;
- ***Write Actions*** – Junta novas ações à presente lista de ações. Caso já exista alguma ação do mesmo tipo este comando irá sobrepor-se à ação, caso contrário a ação será adicionada à lista de ações a tomar.
- ***Stat-Trigger*** – Gera um evento para o controlador caso alguma estatística de determinada entrada de *flow* ultrapasse o limiar definido.
- ***Goto – Table*** – Indica a próxima tabela onde o pacote deverá ser encaminhado. O ID da nova tabela deverá ser superior ao da tabela atual. Esta ação deve ser suportada por todas as tabelas exceto na última.

A figura 9 revela o processo de correspondência de um pacote e o seu tratamento dentro de uma tabela de *flow*. Um pacote, ao entrar numa tabela de *flows* é efetuada uma tentativa de correspondência (*match*). Caso não ocorra correspondência um pacote deverá ser tratado como um *table-miss*. Caso ocorra correspondência um pacote é submetido a um conjunto de ações (*Action Set*) inerentes à entrada de *flow* para o qual o pacote foi correspondido. Num conjunto de ações podem existir diversas ações a executar. Um pacote poderá ser submetido a diversas ações do tipo A*pply-Action* e/ou ainda à instrução *Goto – Table*.

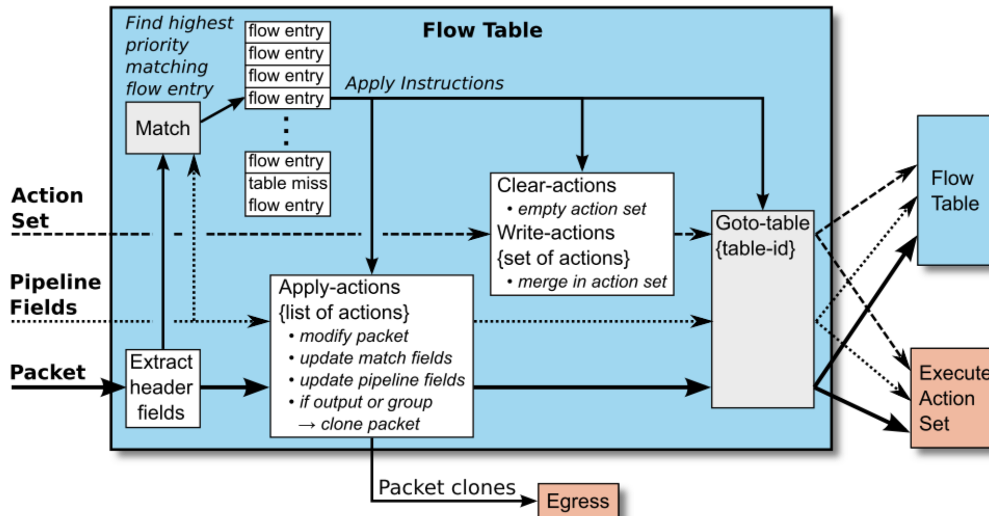

Figura 9 – Correspondência de pacotes e execução de instruções numa tabela de *flow* [19]

2.7.5 Ações

As ações são executadas no âmbito de um conjunto de ações (*Action-set*) presente numa entrada de *flow*. Um conjunto de ações pode ordenar a execução de diversas ações de diversos tipos.

A lista de ações presente numa instrução do tipo *Apply-Actions* possui uma certa ordem de precedências ao nível das ações que podem ser executadas. A lista implica a seguinte precedência entre ações:

1. ***Copy TTL inwards***: Copia o TTL (*Time-to-Live*) do cabeçalho mais interior de um pacote para o cabeçalho mais exterior.
2. ***Pop***: *Remove* uma tag de um pacote;
3. ***Push-MPLS***: Aplica uma tag MPLS (*Multi Protocol Label Switching*) ao pacote;
4. ***Push-PBB***: Aplica uma tag PBB (*Provider Backbone Bridge*) ao pacote;
5. ***Push-VLAN***: Aplica uma tag VLAN (Virtual *Local Area Network*) ao pacote;
6. ***Copy TTL outwards***: Copia o TTL do cabeçalho mais exterior de um pacote para o cabeçalho mais interior.
7. ***Decrement TTL***: Decrementa o TTL de um pacote;
8. ***Set***: Aplica uma ação do tipo *set-field*;
9. **Qos**: Aplica ações de QoS como por exemplo *meter* e *set_queue*;
10. ***Group***: Aplica ações presentes num determinado *group bucket*(ver 2.7.6)
11. ***Output***: Se não existe nenhuma ação de *group* especificada, encaminhar o pacote para a porta de saída especificada pela ação de *output*

As precedências indicadas são referentes à versão 1.5 do OpenFlow. A cada nova versão do protocolo OpenFlow foi introduzida cada vez uma maior complexidade no que diz respeito à quantidade de ações disponíveis a executar através de um flow.

Ao nível do *egress*, uma lista de ações não pode conter uma ação do tipo *output* ou do tipo *group*. Aquando da receção de um pacote a uma tabela de *flows* de *egress,* o pacote já estará encaminhado para determinada porta de saída, daí nenhuma modificação ser permitida no que toca ao fluxo do pacote.

2.7.6 Tabela de Grupos

Uma tabela de grupo (*Group Table*) permite a disponibilização de métodos de encaminhamento adicionais com base na habilidade de um *flow* poder apontar para diferentes tabelas de grupos. A tabela 4 ilustra os campos que caracterizam uma tabela de grupo.

Group Identifier	*Group Type*	*Counters*	*Action Buckets*

Tabela 4 – Características de um *Group Table* [19]

Cada tabela de grupo possui:

- ***Group Identifier*** – número inteiro de 32 bit que identifica uma tabela de grupo num *switch* de forma única;
- ***Group type*** – Determina a semântica do grupo;
- ***Counters*** – Atualiza sempre que um pacote é processado por um grupo;
- ***Action Bucket*** – Representa um conjunto de listas de ações. Dependendo da semântica da tabela de grupos, uma tabela de grupos pode ter várias *Action buckets*.

Relativamente à característica *group type* estão definidos 4 tipos possíveis na norma:

- ***Indirect*** **–** Executa apenas uma lista de ações definida no *Action bucket*.
- ***All*** – Executa todos os *Action buckets* presentes no grupo. Este grupo é frequentemente utilizado para *multicast* ou encaminhamento de *broadcast*. Um pacote ao ser recebido é clonado para cada um dos *Action buckets* de maneira a difundir o pacote para vários *output*s.
- ***Select*** **–** Os pacotes são processados apenas por um *Action bucket* previamente selecionado.
- ***Fast Failover*** **–** Executa os primeiros 5 *Action buckets* que ainda estejam ativos. Cada *Action bucket* está associado a uma determinada porta ou grupo que controla a sua atividade. Caso não existam *Action buckets* ativos os pacotes serão descartados.

2.7.7 Tabelas de Medição

Uma tabela de medição (*meter table*) consiste num conjunto de entradas de medição que possibilitam a hipótese de implementar restrições em determinados *flows* impondo: limites de débito binário, operações de QoS simples e políticas de policiamento de tráfego complexas e detalhadas. Por exemplo, a partir destas tabelas, é possível medir o débito alcançado por cada tipo de pacotes com determinado campo DSCP (*Differentiated Services Code Point*) associado.

Todas as entradas de uma tabela de *flow* medem o débito binário reproduzido pelos pacotes que são correspondidos, desta forma um *switch* pode medir e controlar o débito permitido por cada entrada de *flow* ou conjunto de entradas de *flows*.

Para o efeito, a norma prevê a criação de uma tabela de medição que serve para manter o registo dos pontos de medição presentes num *switch*. Uma entrada de medição (*meter entry*) possui as características identificadas na tabela 5:

Meter Identifier	*Meter Bands*	*Counters*

Tabela 5 – Principais componentes de uma *meter entry* [19]

Cada entrada de medição possui:

- ***Meter identifier*** – número inteiro de 32 bit que identifica de forma única uma entrada de medição;
- ***Meter band*** – Uma lista não ordenada de bandas de medição onde é detalhada a forma como a banda de medição é processada;
- ***Counters*** – Atualizado a cada pacote que é processado pela medição.

Diferentes entradas de *flow* podem estar associadas à mesma banda de medição, a diferentes bandas de medição ou a nenhuma banda. Um pacote pode ser processado por diversos pontos de medição de tráfego enquanto o mesmo é processado pelas tabelas de *flow*.

Cada entrada nesta tabela pode ter várias bandas de medição. Conforme mencionado, as bandas de medição definem a regra a aplicar para que uma medição seja efetuada. A medida pode ser efetuada com base em todos os pacotes de todas as

entradas de *flow*. Por cada pacote, uma entrada da tabela de medição seleciona uma banda de medição com base no débito binário medido, nos valores de medição da banda e na configuração da medição. Um pacote só pode ser processado por uma única banda de medição de uma entrada de medição da tabela de medição. A figura 10 ilustra o acima descrito onde pacotes correspondidos na tabela de *flow* são medidos pelas entradas da tabela de medição (quadrado rosa) onde, com base no débito medido, é selecionada uma determinada banda.

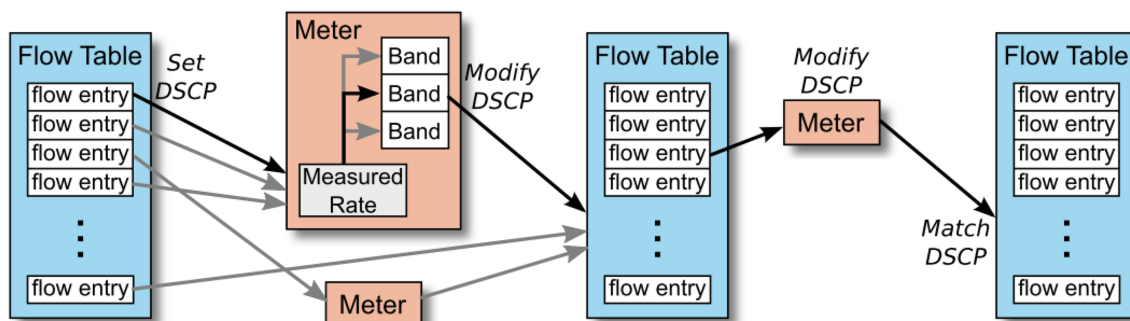

Figura 10 – Fluxo de medição das entradas de medição [19]

Uma banda de medição (*meter band*) é caracterizada pelo seu tipo, pelo débito necessário para aplicar a banda, pela sua granularidade de medição, pelos seus contadores e ainda outros argumentos específicos. A tabela 5 ilustra os principais componentes de uma banda de medição:

Band Type	*Rate*	*Burst*	*Counters*	*Type specific arguments*

Tabela 6 – Principais características de uma banda de medição

Cada banda de medição possui:

- ***Band Type*** – Indica ação despoletada por esta banda caso o débito de determinado *flow* exceda o rate;
- ***Rate*** – débito para o qual a banda de medição cumpre a sua função;
- ***Burst*** – Define a granularidade da banda de medição;
- ***Counters*** – Atualiza sempre que os pacotes são processados pela presente banda;
- ***Type specific arguments*** – Alguns tipos de banda podem ter argumentos específicos.

Estão previstos pela norma dois tipos de banda (*band types*), *drop* que provoca o descarte do pacote caso o débito medido exceda o valor definido pelo campo *rate* e *dscp remark* que aumenta a precedência de *drop* no campo DSCP do cabeçalho IP.

2.7.8 *OpenFlow Channel*

O *OpenFlow Channel* é a designação dada à interface que interliga o *switch* ao controlador *OpenFlow*. Através desta interface o controlador configura, monitoriza e gere os equipamentos da rede recebendo eventos e enviando comandos. Toda a informação *OpenFlow* no sentido *switch*-controlador é designada um *packet-in*

enquanto toda a informação enviada no sentido controlador-*switch* é designado de *packet-out*. Um *switch* pode suportar um ou mais *OpenFlow Channels* para comunicar com vários controladores que partilhem a gestão do *switch*. Todas as mensagens que são enviadas através deste canal devem ser encriptadas pelo protocolo TLS (*Trans*port *Layer Security*).

Na figura 11 surge uma ilustração dos principais componentes de um *OpenFlow switch,* tendo presente, todos os elementos do pipeline de processamento de pacotes e a sua interação com o controlador.

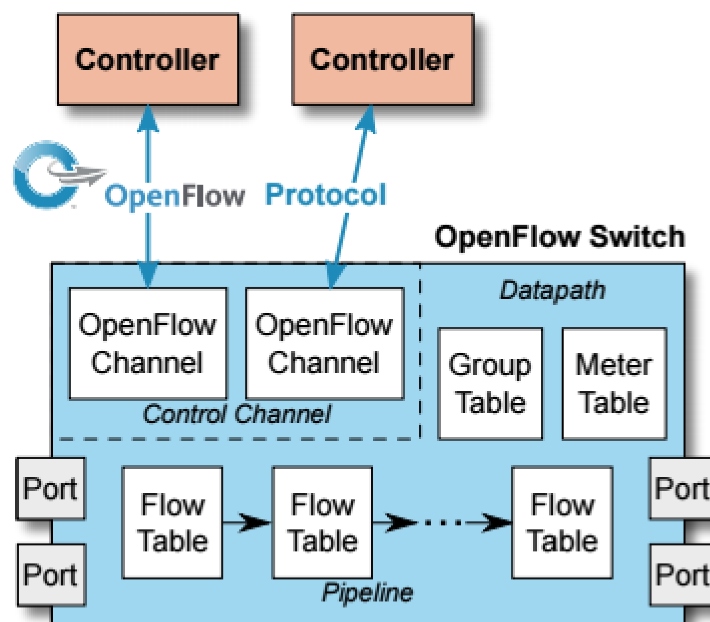

Figura 11 – Componentes principais de um *OpenFlow Switch* [19]

2.7.8.1 Mensagens *OpenFlow*

O protocolo *OpenFlow* suporta 3 tipos de mensagens:

- ***Controller-to-switch*** – Mensagens Iniciadas pelo controlador que são usadas para recolher informações e enviar comandos para o *switch*;
- ***Asynchronous*** – Mensagens iniciadas pelo *switch* com a finalidade de atualizar informações relativas a eventos que ocorram no *switch* ao controlador;
- ***Symmetric*** – Mensagens iniciadas tanto pelo controlador como pelo *switch*. Estas mensagens são sempre enviadas sem solicitação de qualquer parte.

Cada tipo de mensagens possui diversos subtipos que serão descritos nos seus respetivos subcapítulos.

Controller-to-Switch

As mensagens do tipo *controller-to-switch* têm a sua origem no controlador e podem ser dos seguintes subtipos:

- ***Features:*** Solicita identidade e características que o *switch* suporta. Utilizado frequentemente aquando do estabelecimento de um canal de comunicação;
- ***Configuration:*** Solicita e impõe configurações a um *switch*;
- ***Modify-State:*** Adiciona, remove ou modifica entradas em tabelas de *flow*, grupo ou medição, insere ou remove *Action* buckets de grupos ou altera propriedades de uma porta;
- ***Read-State:*** Utilizado para recolher diversas informações do *switch* tais como, configuração atual, estatísticas ou capacidades do *switch*;
- ***Barrier:*** Utilizado para assegurar que as operações foram concluídas;
- ***Role-Request:*** Utilizado para identificar o papel de determinado *OpenFlow Channel*. Um *switch*, aquando da utilização de múltiplos controladores, necessita distinguir qual o controlador mestre e quais os controladores redundantes;
- ***Asynchrnous Configuration:*** Utilizada para criar filtros em *switches* no que toca à receção de mensagens.

Asynchronous

Este tipo de mensagens é originado por um *switch* sem qualquer aviso ao controlador. Estas mensagens normalmente possuem informação sobre súbitas alterações num *switch*.

- ***Packet-in:*** Mensagem que transfere o controlo de um pacote ao controlador. Eventos como a ocorrência de uma falha de correspondência (*table-miss*);
- ***Flow Removed:*** Informa o controlador que determinada entrada da tabela de *flow* foi removida. Mensagens deste tipo são despoletadas quer pela expiração de determinada entrada de *flow* na tabela ou pela ordem explícita de remoção por parte do controlador;
- ***Port-Status:*** Informa o controlador da alteração do estado de uma porta. Inclui especificações sobre o que despoletou essa alteração quer por alteração da configuração da porta ou por falha de sinal na respetiva porta;
- ***Role-Status:*** Informa ao controlador a alteração do seu papel enquanto controlador mestre ou controlador redundante;
- ***Controller-Status:*** Informa controlador caso ocorra alguma alteração do estado no *OpenFlow Channel*. Facilita a comunicação entre controladores caso falhe algum dos canais de comunicação;
- ***Flow-Monitor:*** Comunica-se a alteração de uma tabela de *flow*. Um controlador pode definir um conjunto de regras de monitorização sobre determina tabela num *switch*.

Symmetric

Uma mensagem do tipo *Symmetic* pode surgir sem qualquer solicitação quer por parte do controlador quer por parte do *switch*.

- ***Hello:*** Mensagem trocada após estabelecimento do *OpenFlow Channel*;
- ***Echo:*** Uma mensagem do tipo *echo* espera sempre uma mensagem do tipo *reply*. Esta mensagem é utilizada para verificar conectividade, medições de latência ou banda no *OpenFlow Channel*;

- ***Error:*** Utilizada para reportar a ocorrência de erros. Normalmente utilizada para comunicações de problemas por parte do *switch*;
- ***Experimenter:*** A norma prevê a possibilidade de serem criadas mensagens para efeitos de Testes ou criação de novas funcionalidades.

2.7.8.2 Estabelecimento de *OpenFlow Channel*

De maneira a estabelecer um canal de comunicação *OpenFlow* um *switch* deverá correr um setup inicial e efetuar comunicação através do URI (*Uniform Resource Identifier*) apropriado de maneira a identificar o endereço do controlador, o protocolo e ainda, opcionalmente, o porto de comunicação. Segundo o RFC (*Request For Comments*) 3986 [48], um URI pode assumir a seguinte forma:

- *Protocol:name-or-Address:port*

Protocol indica o protocolo a utilizar, tipicamente utilizam-se protocolos que garantam alguma robustez e segurança como é o caso de protocolos como o TCP e o TLS.

Name-or-Address, indica o endereço IP do controlador ou o nome caso na rede exista algum mecanismo de DNS (*Domain Name System*) que converta o nome do controlador para o seu endereço IP.

Port, indica qual o porto a utilizar pelo *switch*. Na eventualidade do porto mesmo não estar descrito no URI o *switch* deverá assumir como defeito o porto 6653.

A comunicação entre o *switch* e um controlador pode ser efetuada através de várias redes, isto é, o canal de comunicação pode ser estabelecido através da própria rede que faculta serviços (comunicação dentro da banda) ou através de um circuito dedicado (fora da banda). No caso de se optar por uma gestão dentro da banda, o tráfego de serviço e o tráfego de gestão é todo ele rececionado pelas mesmas interfaces do *switch*, neste sentido é necessário que um conjunto de *flows* bem definido possa extrair do processamento *OpenFlow*, o tráfego de comunicação *OpenFlow*.

O estabelecimento de um canal pode ser iniciado tanto pelo controlador como pelo *switch*, tipicamente o *switch* inicia a comunicação sobre uma ligação TCP ou TLS. Após estabelecer essa ligação é efetuada a troca de mensagens *Hello*. Nesta mensagem é enviada a informação da versão *OpenFlow* mais alta que o *switch* suporta. O controlador por sua vez, ao receber uma mensagem de *Hello* responde ao *switch* com uma nova mensagem de *Hello* onde indica a aceitação da versão indicada pelo *switch*. Caso o controlador não suporte essa versão do protocolo *OpenFlow* o controlador envia na mesma uma mensagem Hello com indicação de falha.

Após o correto estabelecimento de uma comunicação *OpenFlow* o controlador envia uma mensagem do tipo *features-request* onde solicita todas as propriedades/características do *switch*.

Todo este processo está ilustrado na figura 12.

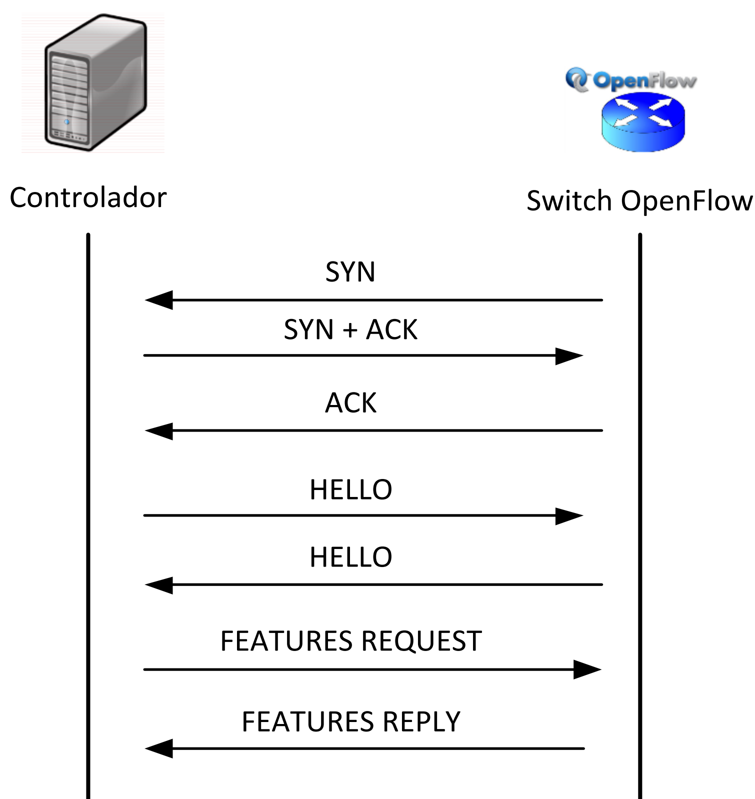

Figura 12 - Estabelecimento de ligação *OpenFlow*

2.7.8.3 Manutenção de *OpenFlow Channel*

A manutenção do canal *OpenFlow* é fundamentalmente gerida pelos protocolos TLS e TCP. Mecanismos como TCP *timeouts* ou TLS *session timeouts* permitem a um *switch* aperceber se da falha da ligação com os respetivos controladores. Outro mecanismo de manutenção da ligação passa pela periódica troca de mensagens *OpenFlow Echo.*

Aquando da falha de um canal *OpenFlow* um *switch* deverá periodicamente tentar reestabelecer a comunicação com o controlador. O intervalo de tempo entre tentativas de restabelecimento de canal *OpenFlow* deverá ser progressivamente mais longo de maneira a não causar qualquer congestionamento na rede.

2.7.8.4 Controladores Múltiplos

Um *switch* pode estabelecer comunicação com um ou mais controladores *OpenFlow*. Ao estabelecer comunicação com múltiplos controladores a robustez da solução SDN aumenta. Idealmente um *switch* nunca deverá estar associado apenas a um controlador Para estabelecer comunicação com múltiplos controladores, um *switch*, apenas necessita de criar com sucesso *OpenFlow Channels* dedicados a cada um dos controladores para conseguir comunicar com os respetivos controladores.

O estabelecimento de ligações a múltiplos controladores implica que exista um mecanismo de comunicação entre controladores via interfaces *east/westbound*. Neste sentido um *switch* apenas deverá conceder direitos de escrita apenas ao controlador que

ficar definido como controlador mestre, enquanto os restantes controladores deverão ser definidos como controladores escravos.

A decisão de qual o controlador que deve assumir funções de controlador mestre compete aos mecanismos de gestão e sincronismo entre controladores. Um *switch*, após criar os canais de comunicações com os múltiplos controladores, considera todos os controladores como iguais. O controlador que for selecionado como controlador mestre deverá enviar uma mensagem ao *switch,* no sentido de requerer uma atualização do seu estado, de maneira a que este, passe a ser de controlador mestre. Um *switch* ao atualizar o estado de um controlador para controlador mestre atribuirá o estado de *slave* aos restantes controladores. Um *switch* apenas pode ter um controlador mestre. Todavia um controlador pode sempre solicitar a alteração do seu estado. Um controlador mestre pode executar qualquer ação sobre um *switch,* enquanto um controlador escravo apenas poderá enviar comandos de leitura, deste modo um *switch* fica protegido de cenários de múltiplas escritas.

2.8 Limitações do protocolo *OpenFlow*

O protocolo *OpenFlow*, embora bastante inovador e completo tem sido alvo de diversas criticas, [22-24] que expõem algumas das fragilidades deste protocolo.As limitações estão, fundamentalmente, relacionadas com problemas de escalabilidade no que toca ao:

- Número de equipamentos que um controlador gere;
- Número de entradas de *flow* suportáveis por um *switch*.

Em redes com elevado número de equipamentos verifica-se a necessidade de um controlador manter uma ligação TCP ou TLS individual com cada um dos equipamentos, de maneira a poder efetuar todas as operações *OpenFlow* previstas na norma.

Este aspeto revela que um controlador *OpenFlow* pode ser limitado pela performance do CPU e pelas sessões TCP que consegue manter. Uma primeira abordagem na resolução desta limitação passaria pela colocação de controladores adicionais na rede, na condição de reduzir o número de equipamentos por controlador diminuindo o número de ligações *OpenFlow* a manter por cada controlador. A adição de controladores implica que tenha de existir um mecanismo de alto nível que seja capaz de manter a informação coerente e consistente pelos diversos controladores.

Relativamente ao número de *flows* suportados por um *switch*, este problema tem sido endereçado com bastante atenção por parte da ONF. Verifica-se que as sucessivas novas versões do protocolo tem melhorado de forma global introduzindo novas funcionalidades. Entre estas destaca-se a possibilidade poderem existir *switches* com múltiplas tabelas de *flow*. Na figura 13 é possível identificar os progressos e adições de novas funcionalidades a cada nova versão do protocolo *OpenFlow*.

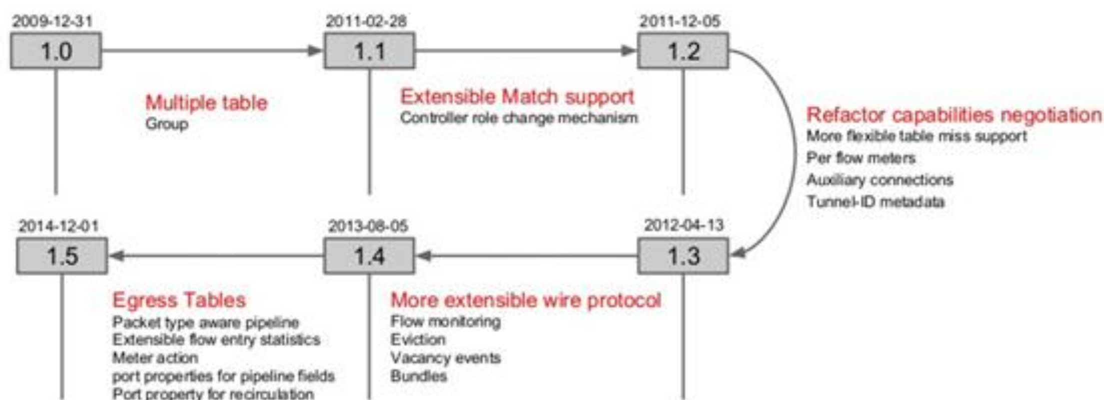

Figura 13 - Evolução do Protocolo *OpenFlow* a cada Versão [21]

Switches modernos atualmente têm capacidade de suportar FIB (*Fowarding Information Base*) com capacidade de gerir tabelas de 64k a 512k entradas permitindo desta forma dar alguma margem de manobra adicional a administradores de rede na implementação das suas políticas de encaminhamento e qualidade de serviço [20]. No anexo A é possível observar com maior detalhe as características de cada versão.

2.9 Aplicações SDN

Atualmente, a implementação de aplicações sobre redes de telecomunicações é bastante diversa. Existem aplicações de monitorização, gestão, balanceadores de tráfego e *Firewall*s que permitem melhorar o desempenho de uma rede, tanto do ponto de vista da sua gestão, como do ponto de vista de serviço. Aplicações como Cacti, ProVision, OpenNMS são aplicações baseadas em protocolos específicos tais como SNMP e NETCONF. Este tipo de aplicações são instaladas em servidores que necessitam de estabelecer comunicação com toda a rede tal como um controlador.

Numa rede tradicional a introdução de aplicações implica a compatibilidade da aplicação com diversos equipamentos de diferentes tipos e fabricantes para além de que cada aplicação consome recursos individuais em cada equipamento traduzindo-se numa maior quantidade de tráfego de gestão na rede. Numa rede SDN o desenvolvimento de aplicações torna-se algo bastante simples e acessível através do uso da *northbound* interface.

Conforme já mencionado a *northbound* interface disponibiliza uma API, o que permite o desenvolvimento de aplicações SDN sobre linguagens bastante utilizadas atualmente como *JAVA*, C, Python, entre outras. Tal permite que o processo de inovação sobre uma rede seja feito de uma forma simples e ao mais alto nível possível permitindo a programadores terem acesso a todo o tipo de dados presentes na rede podendo desenvolver o mais diverso tipo de aplicações independentemente do tipo de equipamento ou fabricante.

Neste âmbito a HP tem desenvolvido um trabalho bastante extenso dando já os primeiros paços na abertura de portas à inovação numa rede SDN. Segundo [25], a HP constituiu a primeira loja de aplicações SDN oferecendo um diverso leque de serviços e funcionalidades. Naturalmente os fabricantes neste momento concentram esforços em marcar uma posição no mercado e a HP não é exceção, a loja de aplicações lançada pela HP apenas é compatível com o controlador HP VAN SDN. Tal deve-se ao facto de não

existir uma norma que normalize a *northbound* interface dos diversos controladores disponíveis no mercado.

2.10 *Network Functions Virtualization*

As NFV (*Network Functions Virtualization*) são um conceito bastante complementar às redes SDN. As NFV têm como foco principal a otimização dos serviços oferecidos pela rede através da virtualização de funções onde por vezes é utilizado hardware específico para o efeito como por exemplo, *Firewall*s, gestão DNS, caching, entre outras funcionalidades. [34]

O conceito de uma rede SDN aliado às diversas NFV disponíveis traduzem-se em grandes vantagens do ponto de vista operacional e também em termos de custos da própria rede. Grande parte dos controladores SDN já implementam diversas funções virtualizando listas de acesso, gestão DNS, *Firewall*s, *Load Balancers* entre outras.

2.11 Segurança em SDN

Numa rede SDN a comunicação com os equipamentos converge para uma única entidade denominada de controlador.

Em termos de segurança este facto evidência que todas as comunicações com interação com o controlador tenham de ser consideradas como críticas dado que há mínima fragilidade seja possível que toda a rede SDN e seus serviços sejam comprometidos.

Analisando uma rede SDN típica é possível estabelecer como principais ameaças os seguintes pontos:

1. Forjamento de *Flow*s;
2. Vulnerabilidades nos *Switches;*
3. Ataques ao plano de controlo;
4. Exploração de vulnerabilidades do controlador;
5. Falta de mecanismos que assegurem a autenticidade das comunicações com o controlador;
6. Ataques às estações administrativas;
7. Falta de meios para estudo forense da falha do sistema e contenção de impactos.

A figura 14 expõe a localização das principais ameaças mencionadas:

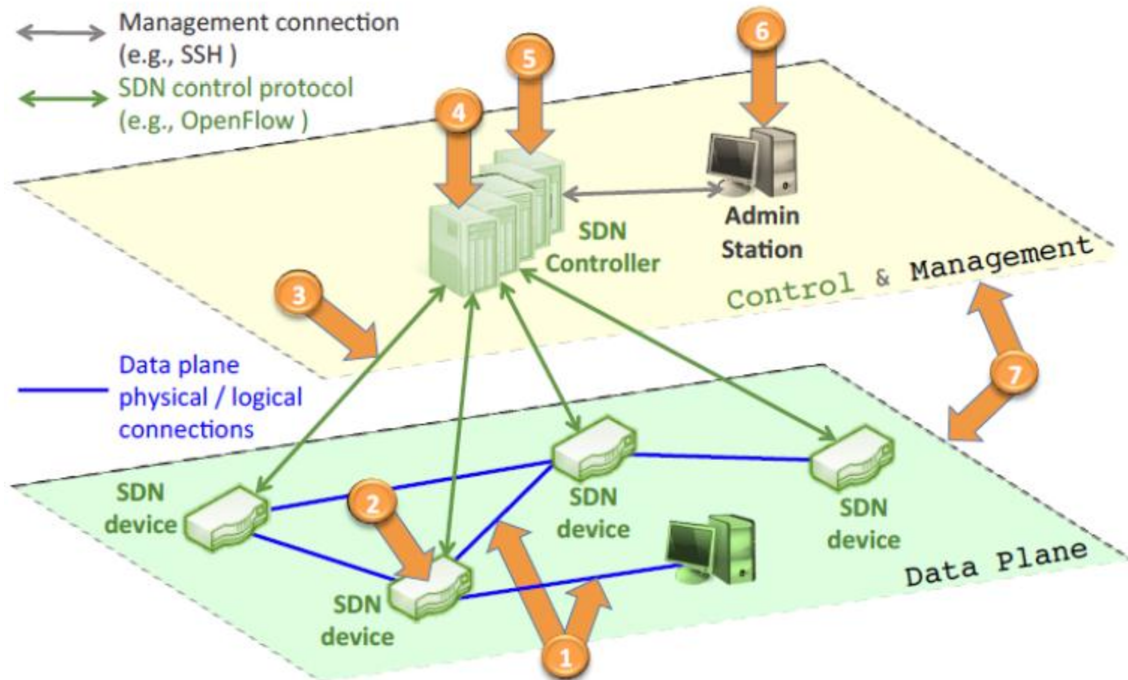

Figura 14 – Exposição das principais ameaças a uma rede SDN. [38]

Aquando do dimensionamento de uma rede SDN a segurança torna-se um elemento critico pelo que a segurança nesta rede terá de assumir um papel muito importante. De maneira a aumentar a robustez de uma rede SDN é necessário que sejam implementadas diversas políticas de segurança nos diferentes planos de dados, de controlo e de aplicação. Essas medidas podem ser:

- Implementação de comunicação TLS em todas as interfaces do controlador;
- Implementação de replicação de controladores para cenários de falha de um servidor;
- Monitorização ativa das interfaces de controlador para mitigação de ataques de DDOS;
- Criação de logs nos vários planos da rede SDN para controlo e rastreamento de acesso;
- Aplicação de medidas de contenção de danos;
- Alteração de passwords de fábrica.

2.12 Implementação de casos reais

Atualmente as SDN ainda não encontram uma posição sólida no mercado sendo ainda uma tecnologia em fase de maturação existindo diversas concentrações e eventos [35], [36] no sentido de promover este novo conceito para as redes de internet. Até ao momento esta tecnologia tem sido aplicada fundamentalmente em *datacenters* que suportam redes locais de alto débito.

Muitos operadores ainda estão um pouco céticos à ideia de centralizar a inteligência da sua rede em controladores, o que levanta diversas questões no que toca à robustez e segurança do sistema.

Em 2011, empresas como a *Google*, *Facebook*, *Microsoft*, *Deutsche Telekom*, *Verizon* e *Yahoo!* [5] tomaram iniciativa e criaram a ONF que é uma organização sem fins lucrativos dedicada em repensar o atual panorama das redes de computadores, criando sinergias para trazer para o mercado normas e soluções baseadas em SDN. Esta organização, à semelhança do projeto *Linux*, promove um conceito de rede de fonte aberta, onde é possível utilizar um determinado SO num dado equipamento e gerir a rede com um software específico também designado por controlador.

Nos seguintes subcapítulos será dado a conhecer algumas das implementações SDN de empresas como a *Google*, *Facebook* e ainda *Microsoft*.

2.12.1 Google

Grandes redes requerem enorme inovação, no caso do Google desde 2004 que os principais fabricantes não são capazes de satisfazer as exigências de tráfego que os milhares de servidores *Google* geram. Em [6], apresentam-se os principais de desafios que a empresa se deparou no sentido de desenvolver soluções que agilizassem o fluxo de tráfego bem como a implementação dessas soluções. Nele são identificadas as principais características da rede SDN desenvolvida tais como a utilização de topologias eficientes, a redução de complexidade da rede, utilizando equipamentos à base de *merchant silicon* [7] (equipamentos com circuitos integrados programáveis) e a criação de um controlador SDN que fosse ao encontro das necessidades do *datacenter*.

2.12.2 Facebook

O *Facebook*, a maior rede social do mundo conta com mais de 1,23 biliões de contas registadas. É sem dúvida um dos maiores sucessos da internet e como tal a sua presença no mercado é dominadora, através das suas constantes ações inovadoras e introduções de novas formas de partilhar conteúdos nos mais diversos formatos.

Ao nível da sua rede o *Facebook* necessitou de se reinventar dado os equipamentos disponíveis no mercado não serem capazes de suportar os enormes volumes de tráfego gerados pelos seus servidores. Em parceria com o OCP (*Open Computer Project*) foram redesenhados os TOR (*Top-Of-Rack*) *switches,* tanto a nível de hardware, como ao nível de software. O objetivo passou pela criação de um equipamento com capacidade de comutar tráfego na ordem dos Tbps possibilitando ao mesmo tempo a utilização de um qualquer SO [8].

Através desta parceria foram desenvolvidos equipamentos como o *Wedge* e o *6-Pack*, ilustrado nas figuras 15 e 16. O *Wedge* surge como um dos primeiros *swtiches* de alta capacidade (comutação na ordem dos Gbps) que introduz a possibilidade de se poder aplicar um qualquer SO, introduzindo pela primeira vez o conceito de *Bare-Metal switch* na indústria das telecomunicações. Ao nível de SO o *Facebook* desenvolveu uma distribuição *Linux* denominada de FBOSS otimizada para os seus propósitos.

Figura 15 – *Wedge switch [9]*

Relativamente ao *6-Pack*, é um equipamento desenvolvido tendo em conta escalabilidade da rede sendo ao mesmo tempo um equipamento simples e de alta capacidade constituído por apenas 6 cartas com interfaces 16x40Gbps e duas controladoras com capacidade de comutar cerca de 1,28 a 2,56Tbps dependendo da configuração.

Figura 16 – TOR *Switch* "6-Pack" desenvolvido pelo *Facebook*. [9]

2.12.3 Microsoft - Azure

A implementação SDN da *Microsoft* surge através da criação da rede *Azure* que se baseia numa rede que oferece os mais variados serviços de computação em *cloud*. A rede *Azure* pretende oferecer aos seus clientes uma flexibilidade dos seus serviços com escala mundial, permitindo um maior alcance e melhor qualidade de serviço a todos os serviços *Azure*.

Seguindo as linhas das definições da ONF a *Microsoft* implementou a sua rede através da separação da gestão do controlo e da comutação nos diversos equipamentos proprietários. Na figura 17, verifica-se a cisão entre os planos de gestão, de controlo e de dados ilustrado numa *keynote* apresentada por Mark Russinovich (CTO da Azure) na ONS (*Open Networking Summit*) em 2015 distribuindo o controlo da rede em diversos pontos e otimizando os recursos de hardware.

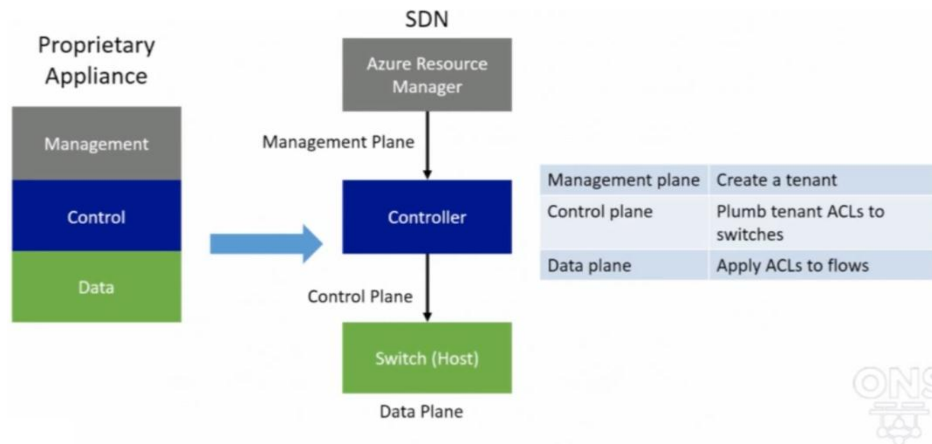

Figura 17 – ONS 2015: Mark Russinovich

Na referida apresentação são mencionadas a construção de s*mart* NIC (*Network Controller Interfaces*) com capacidades especiais que permitem aliviar carga das CPU (*Central Procesing Unit*) transpondo para o *hard*ware funções como encriptação de dados e qualidade de serviço. Destaca-se ainda a utilização de FPGA (*Field Programmable Gate Array*) reconfiguráveis que facultam uma maior flexibilidade ao hardware.

Com esta primeira abordagem às SDN verifica-se que o conceito é bastante abrangente obrigando fabricantes e operadores a repensarem por completo as redes, tanto ao nível de equipamentos, como a própria forma como se opera e se mantem a rede. As SDN permitem acelerar o desenvolvimento de uma rede reduzindo imenso os custos associados com a instalação de equipamento, sendo a sua aplicabilidade bastante útil em ambientes de elevada densidade de nós como em *datacenters*.

Capítulo 3

Desenvolvimento de Ambiente de Testes

No sentido de desenvolver uma prova de conceito e exploração das diversas capacidades que uma rede SDN pode oferecer optou-se por criar uma topologia totalmente gerida por um controlador SDN. Para tal é necessário definir uma topologia, selecionar um controlador e ainda um IDE (*Integrated Development Environment*) de maneira a programar aplicações no sentido de utilizar as funcionalidades oferecidas pelo controlador.

3.1 Ambiente de Testes

O ambiente de testes será construído exclusivamente sobre ambiente virtual através de um PC Windows 7 a 64-bit com processador *Intel Core* i5-2410 com 8 GB de RAM utilizando o Oracle Virtual Box. A nível virtual este ambiente é constituído pela utilização de 2 VM (*Virtual Machine*) com recursos de memória e processamento próprio criadas a partir da ferramenta Oracle Virtual Box. Na figura 18 é possível identificar a topologia criada que é constituída por 2 VMs (*Mininet* – VM e *Floodlight* – VM). Ambas as VMs possuem ligação a uma rede *layer* 2 virtual que estabelece a ligação entre as duas máquinas virtuais com a máquina física.

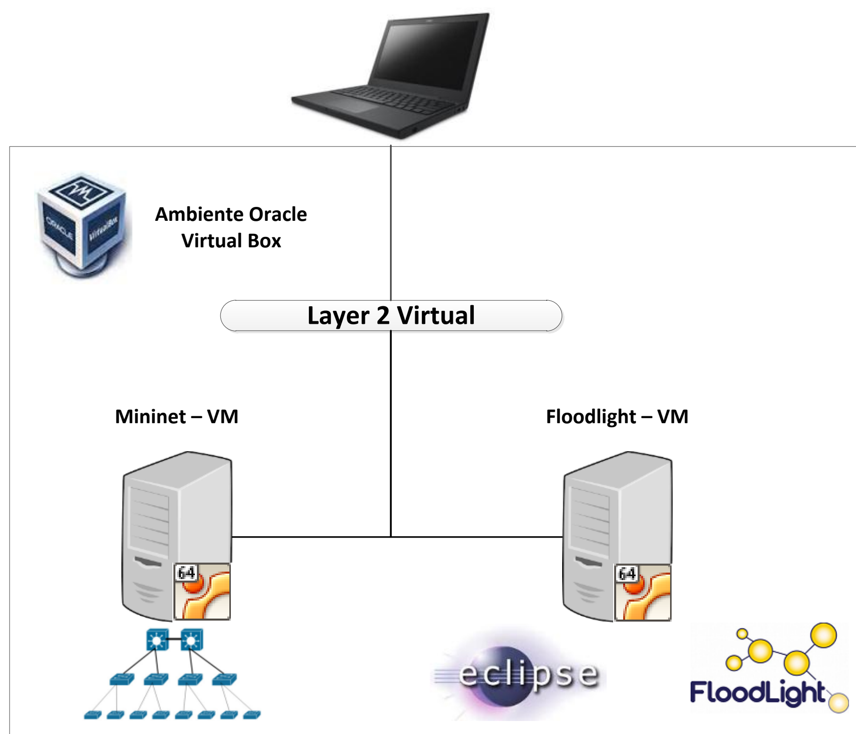

Figura 18 – Descrição da Topologia do Ambiente de Testes

As máquinas virtuais possuem os seguintes recursos dedicados:

***Mininet* – VM**

- Memória:512 *Mega Bytes* RAM
- Processamento: 1 *Core*
- Sistema Operativo: *Ubuntu* 14.04

***Floodlight*– VM**

- Memória: 2 *Giga Bytes* RAM
- Processamento: 2 *Cores*
- Sistema Operativo: *Ubuntu* 14.04

Na *Mininet* – VM será simulada uma topologia com determinado número de *switches* e *Hosts* a definir consoante o cenário pretendido recorrendo ao software *Mininet*. No subcapítulo 3.2 será apresentado o software *Mininet* e todas as suas funcionalidades.

Na *Floodlight* – VM será utilizado o IDE Eclipse para correr uma versão personalizada do controlador *Floodlight*. A VM foi extraída a partir da página oficial do controlador *Floodlight* em [31], enquanto o código fonte do controlador mais atual foi extraído de [40]. O processo de seleção do controlador e a descrição do controlador *Floodlight* encontra-se explicado no subcapítulo 3.3.

Através da *layer* 2 virtual fluirá todo o tipo de comunicação da *northbound* interface e da *southbound* interface, isto é, toda a comunicação entre os diferentes planos da arquitetura SDN fluirá sempre através desta *layer* 2. A comunicação entre o plano de dados e o plano de controlo será abordada com detalhe no subcapítulo 3.4 enquanto entre o plano de controlo e o plano de gestão será aprofundada no subcapítulo 3.5.

Para efeitos de desenvolvimento do plano de gestão, foi instalado o IDE Eclipse onde a partir do qual serão desenvolvidas aplicações programadas em linguagem *Java* utilizando a *Java* API do controlador.

3.2 Seleção de Componente Física

A topologia selecionada para a criação do ambiente de testes passa pela utilização do *Mininet* (software de fonte aberta) que permite emular uma topologia virtual com determinado número de *Hosts* e determinado número de *switches*. Através do *website* [14] é possível efetuar o download de uma imagem virtual que possui todo o software pré-instalado.

Este software torna-se bastante útil dado oferecer bastantes funcionalidades que permitem colocar à prova as aplicações desenvolvidas. É possível utilizar ferramentas como o *iperf*, *ping*, *traceroute* ou até colocar portas up/down em qualquer *Host* ou *switch*. O *Mininet* é agnóstico relativamente ao controlador utilizado desde que o mesmo seja compatível com a versão 1.3 do protocolo *OpenFlow*. O suporte a versões *OpenFlow* mais recentes (1.4 e 1.5) até à data ainda não é suportado pelo *Mininet*. Através deste software também é possível emular *switches* de diversos fabricantes, nos

cenários a apresentar serão apenas utilizados *switches* com software *Open vSwitch* [27] que se trata de um software já disponibilizado pelo *Mininet*.

O *Mininet* trata-se então de um software especializado em desenvolver uma topologia de rede de forma instantânea e compatível com o protocolo *OpenFlow*, podendo desta forma comunicar com qualquer controlador SDN que suporte tal protocolo *southbound*.

3.2.1 Construção de Topologia Virtual

Com o *Mininet* é dada a oportunidade de implementar várias topologias compostas por *switches* e *Hosts* algumas delas já tipificadas como topologias lineares, árvores, Torus [30], entre outras. Adicionalmente é também possível implementar uma topologia personalizável através da criação de um *script python* que definirá a topologia. Na figura 19 surge o código fonte de um script que define uma topologia.

```
from mininet.topo import Topo

class DPTopo( Topo ):

    def __init__( self ):

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's1' )
        middleSwitch1 = self.addSwitch( 's2' )
        middleSwitch2 = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, middleSwitch1 )
        self.addLink( leftSwitch, middleSwitch2 )
        self.addLink( middleSwitch1, rightSwitch )
        self.addLink( middleSwitch2, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'DPTopo': ( lambda: DPTopo() ) }
```

Figura 19 – *Script* que define uma topologia personalizada.

A topologia correspondente ao código apresentado na figura 19 encontra-se ilustrada na figura 20.

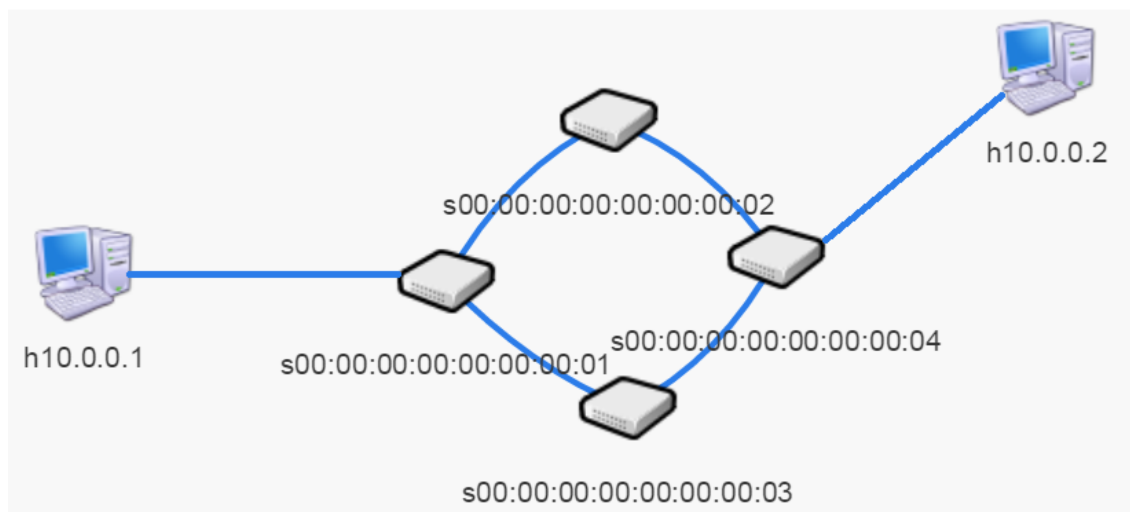

Figura 20 – Ilustração da topologia apresentada.

3.3 Seleção de Controlador SDN

Como já referido, a seleção de um controlador reside fundamentalmente nas características que o mesmo oferece ao nível das APIs fornecidas, dos protocolos suportados, da capacidade de nós SDN e *flows* que consegue suportar etc. Foram efetuados vários testes sobre 3 controladores SDN nomeadamente ODL (*Open Day Light)*, *Floodlight*, e ainda o HP VAN SDN. Após explorar as diferentes características de cada controlador e estabelecer a comunicação entre o controlador e a topologia *Mininet* com sucesso optou-se por utilizar o *Floodlight* dado possuir uma melhor documentação e uma maior estabilidade face aos restantes.

Para o efeito foi criada uma nova imagem virtual com a distribuição *Linux Ubuntu* onde foi instalado o controlador *Floodlight*. Este controlador possui a capacidade de comunicar com equipamentos através de vários protocolos dos quais se destaca a capacidade de comunicar via *OpenFlow* versão 1.3, compatível com a topologia *Mininet*.

3.3.1 Arquitetura do Controlador *Floodlight*

A figura 21 ilustra a arquitetura do controlador *Floodlight*. A figura encontra-se dividida de maneira a ilustrar os serviços core do controlador *Floodlight* que tratam de toda a recolha de dados da rede SDN disponibilizando 2 APIs que podem ser utilizadas de forma completamente não relacionada de maneira a permitir uma melhor facilidade a um programador aceder a dados na rede.

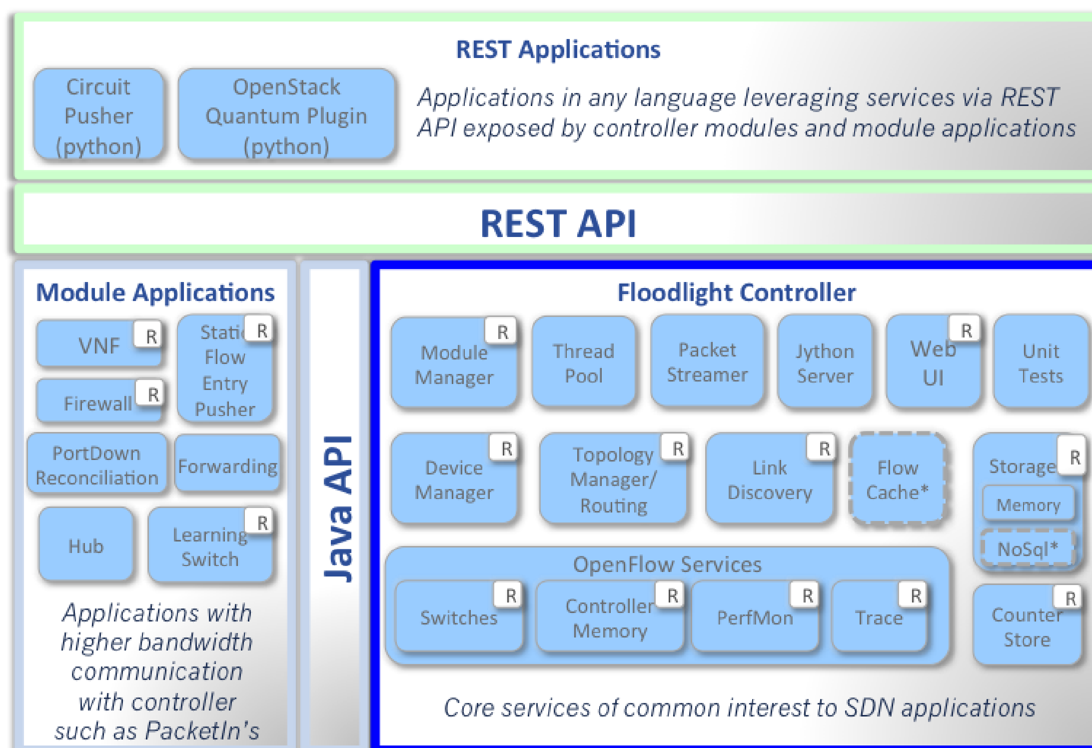

Figura 21 – Arquitetura do Controlador *Floodlight* [42]

Sendo o *Floodlight* uma plataforma de fonte aberta por si só, já oferece a possibilidade de um programador fazer as alterações que entender no código do controlador, ao mesmo tempo que permite a análise de todos os módulos uma vez que existe acesso a todas as linhas de código que definem o software.

As duas APIs possuem propósitos diferentes e tornam-se especializadas para determinado tipo de aplicações. A *Java* API revela ser mais especializada em aplicações *Java* que necessitem de uma grande disponibilidade de dados do controlador e recursos do controlador, neste sentido a *Java* API está definida para que sejam criadas funcionalidades que atuem em paralelo com o controlador. Aplicações como *Firewalls*, *MAC Learning*, Gestão de *flows* constituem alguns exemplos do uso da *Java* API. De maneira a oferecer uma maior liberdade no desenvolvimento de aplicações a um programador, a REST API oferece a possibilidade de se programar em qualquer linguagem no sentido de se aceder aos dados que o controlador pode oferecer. A REST API baseia-se no fornecimento de informação com base em pedidos http. Estes pedidos HTTP são respondidos em formatações de texto bastante utilizadas como XML e JSON. É através do porto TCP 8080 que os pedidos REST podem ser aceites com sucesso.

Tanto a *Java* API como a REST API podem ser trabalhadas para disponibilizar uma funcionalidade que hoje ainda não é suportada pelo controlador. A *Java* API encontra-se disponível em [28], enquanto a REST API encontra-se detalhada em [29]. A *Java* API pode ainda ser utilizada no sentido de criar novas aplicações que permite o acesso a dados do controlador através da REST API, ou seja, a *Java* API é a base da REST API e através dela é possível criar novas funcionalidades na REST API.

3.3.2 Serviços *Floodlight*

De entre os serviços core oferecidos pelo controlador destaca-se a existência de uma interface *web* (WEB UI, *User Interface*) que permite a um utilizador ter acesso a várias informações como por exemplo os serviços que estão implementados, a topologia de rede, os *Hosts* e *switches* presentes na rede.

A figura 22 representa a *view Dashboard* da plataforma *Floodlight* onde é possível identificar alguns traços gerais da rede e do controlador como por exemplo os modulos e os equipamentos presentes (*switches* e *Hosts*)

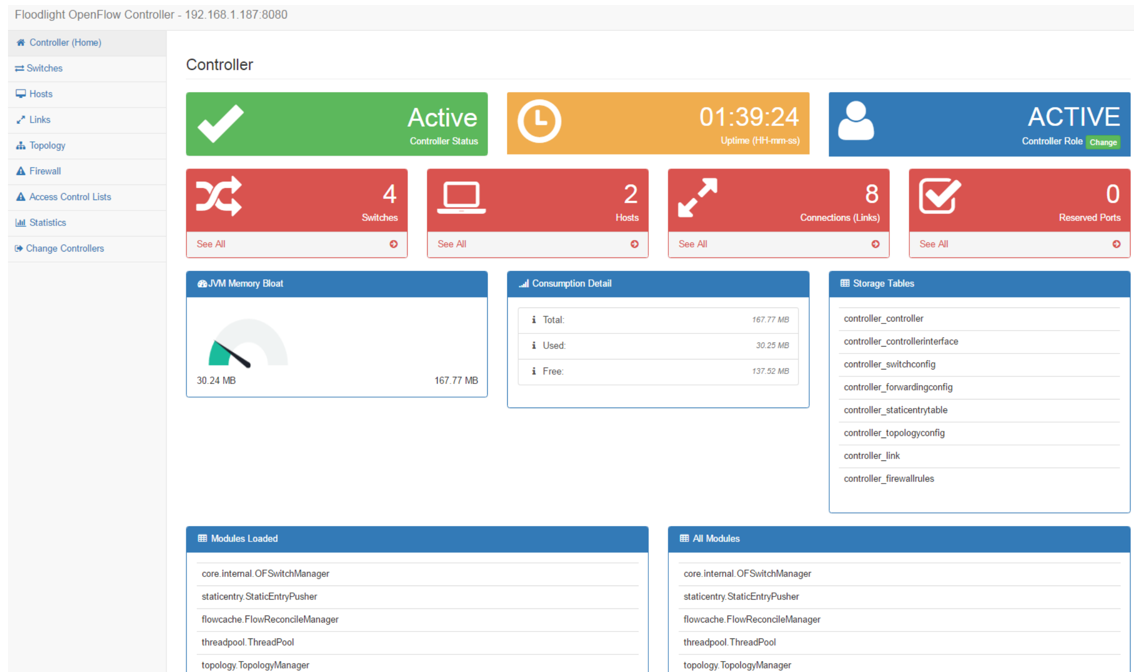

Figura 22 – Interface Web do controlador *Floodlight*, *Dashboard*

A figura 23 ilustra a *view topology* do controlador *Floodlight*. Esta *view* é dinâmica e altera com a entrada de novos nós na rede, *Hosts* ou *switches*.

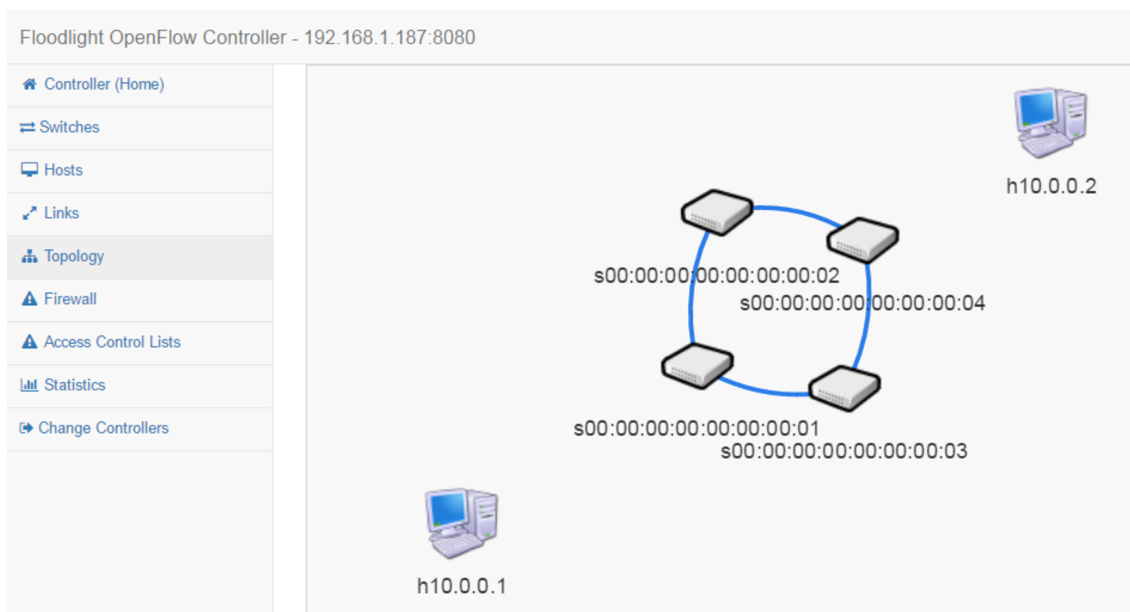

Figura 23 – Interface Web do controlador *Floodlight*, *Topology*

Na figura 24 é possível verificar a *view Switches* que recolhe informações sobre determinado *switch*. Nesta figura verifica-se uma breve descrição do *switch* tal como tabelas relativas a dados estatísticos das portas e *flows* do *switch*.

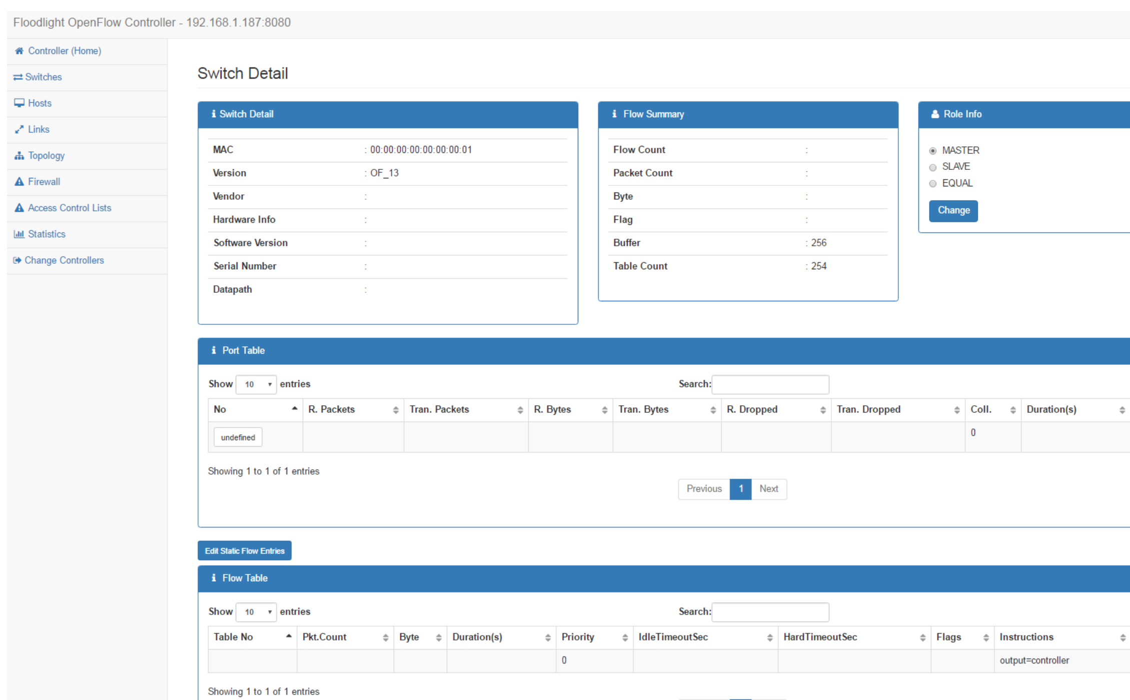

Figura 24 – Interface Web do controlador *Floodlight*, *Switches*

Para além da interface web destaca-se também o serviço *Link Discovery* que este é o serviço responsável pelo conhecimento da topologia da rede *OpenFlow*. Através deste serviço, o controlador ordena os diferentes *switches* a emitirem pacotes LLDP (*Link Layer Discovery Protocol*) e BDDP (*Broadcast Domain Discovery Protocol*) no sentido de apurar os endereços MAC da rede e descobrir as diferentes ligações que existem entre *switches*. O serviço *Topology Service* acaba por recolher dados do serviço

Link Discovery para que seja possível ao controlador otimizar os fluxos de tráfego pelo caminho mais curto, ao mesmo tempo que mantem a noção da rede estando alerta para qualquer acontecimento de eventuais falhas de ligação nos *switches*. O algoritmo aplicado para cálculo dos melhores caminhos de encaminhamento de tráfego é o algoritmo *Dijkstra*.

Outro serviço que possui um papel bastante ativo é o *Device Manager*. Este serviço tem como função comunicar periodicamente com os diferentes equipamentos no sentido de assegurar que a conectividade dos equipamentos com o controlador se mantem e ainda colher diversos dados inerentes a cada equipamento tais como: número de portos, configuração de portos e estado dos portos.

É possível verificar com maior detalhe os restantes serviços através de [45].

3.4 Comunicação *Southbound*

Tendo em conta as características do controlador selecionado e do emulador de rede selecionado (*Mininet*) optou-se por utilizar o protocolo *OpenFlow* como meio de comunicação *southbound*, nomeadamente a versão 1.3 do protocolo *OpenFlow*.

Após arranque do controlador e definição da topologia *Mininet* é iniciado um processo de estabelecimento de comunicação entre os *switches* e o controlador. Todos os *switches* definidos pelo *Mininet* possuem na sua configuração o endereço IP e o porto TCP destino para o qual o controlador está à escuta mensagens *OpenFlow*. No caso concreto do controlador o porto à escuta será o porto TCP 6653.

Na figura 25 encontra-se ilustrada a topologia utilizada para testar a comunicação *OpenFlow* com o controlador. A topologia emulada pelo *Mininet* é composta por 2 *Hosts* (H1 e H2) e 4 *switches* interligados entre eles (S1, S2, S3 e S4) tal como a figura sugere. Estes 4 *switches* possuem adicionalmente uma interface *out-of-band* cuja finalidade é exclusiva à comunicação via protocolo *OpenFlow*, todo o tráfego desta rede não irá fluir por estas interfaces. Aquando da necessidade de comunicar com o controlador, o *Minine,* atribuí um porto TCP para cada *OpenFlow Channel*. Neste cenário em particular onde apenas temos um controlador, os *switches* apenas deverão ter apenas 1 porto TCP atribuído. No caso de falha ou quebra da ligação TCP pode ser atribuído um novo porto TCP pelo *Mininet*.

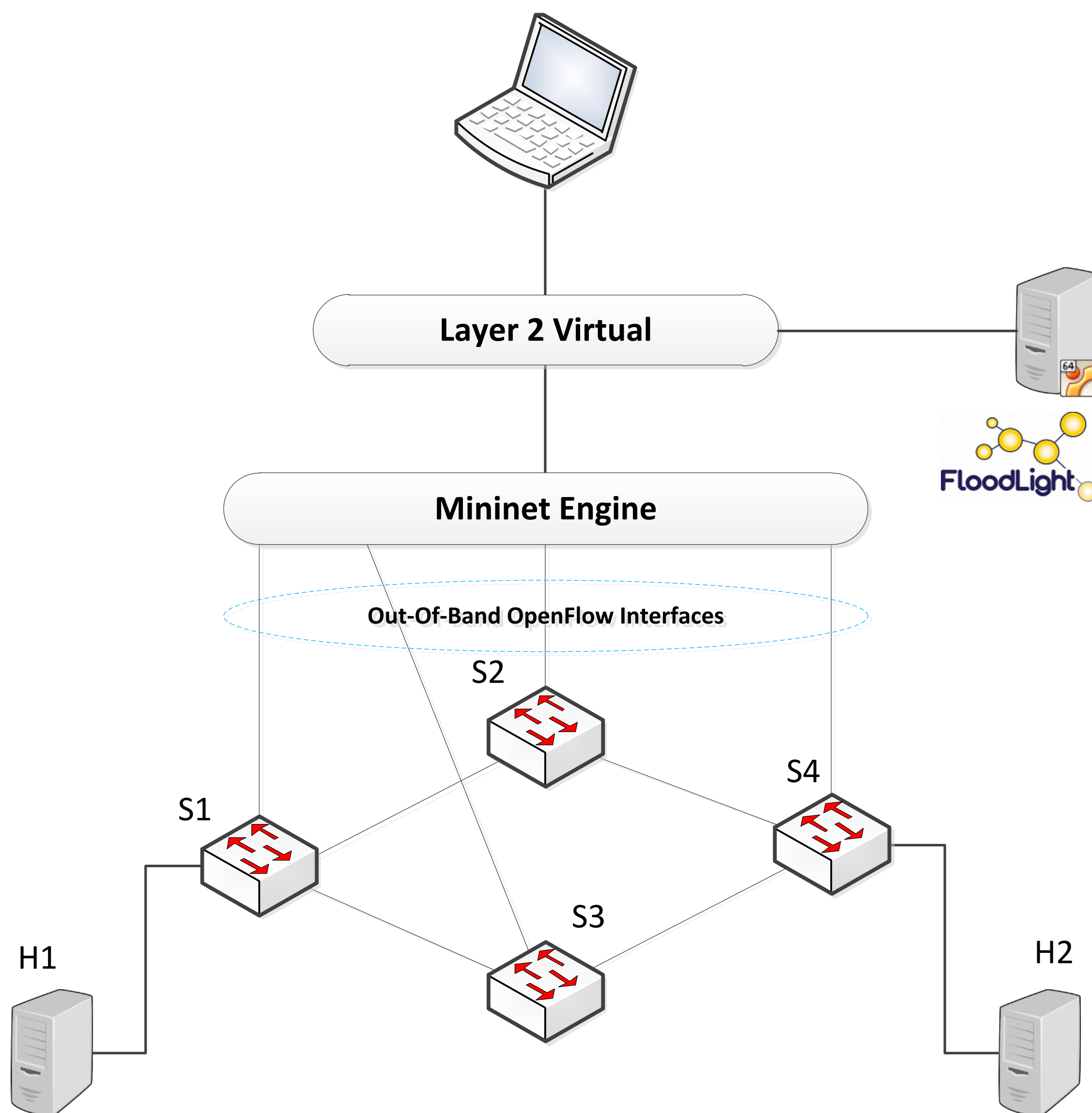

Figura 25 – Topologia Mininet

3.4.1 Estabelecimento de comunicação *OpenFlow*

De maneira a fornecer uma melhor análise do estabelecimento da comunicação *switch*-controlador via *OpenFlow* foi efetuada uma captura de tráfego no porto *ethernet* da *Floodlight* – VM, deste modo é possível verificar todas as interações *OpenFlow* do controlador. A máquina *Mininet* possui o IP 192.168.1.161 enquanto a máquina *Floodlight* possui o IP 192.168.1.187.

Na figura 26, após a leitura do conteúdo dos pacotes *OpenFlow* podemos identificar que o porto TCP 49774 terá sido atribuído pelo *Mininet* ao *Switch* S1.

Após o estabelecimento do *OpenFlow Channel*, verifica-se a troca de mensagens diversas (mensagens *OpenFlow*) como por exemplo: *OFPT_HELLO*, *OFPT_FEATURES_REQUEST*, *OFPT_FEATURES_REPLY* e ainda *OFPT_PORT_STATUS*.

Source	Destination	Protocol	Length	Info
192.168.1.161	192.168.1.187	TCP	74	49774 → 6653 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3990658 TSe
192.168.1.187	192.168.1.161	TCP	74	6653 → 49774 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=
192.168.1.161	192.168.1.187	TCP	66	49774 → 6653 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=3990659 TSecr=3990651
192.168.1.187	192.168.1.161	OpenFlow	74	Type: OFPT_HELLO
192.168.1.161	192.168.1.187	TCP	66	49774 → 6653 [ACK] Seq=1 Ack=9 Win=29696 Len=0 TSval=3990664 TSecr=3990656
192.168.1.161	192.168.1.187	OpenFlow	82	Type: OFPT_HELLO
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=9 Ack=17 Win=29056 Len=0 TSval=3990660 TSecr=3990668
192.168.1.187	192.168.1.161	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
192.168.1.161	192.168.1.187	OpenFlow	98	Type: OFPT_FEATURES_REPLY
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=17 Ack=49 Win=29056 Len=0 TSval=3990677 TSecr=3990685
192.168.1.161	192.168.1.187	OpenFlow	146	Type: OFPT_PORT_STATUS
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=17 Ack=129 Win=29056 Len=0 TSval=3990679 TSecr=3990687
192.168.1.161	192.168.1.187	OpenFlow	146	Type: OFPT_PORT_STATUS
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=17 Ack=209 Win=29056 Len=0 TSval=3990680 TSecr=3990688
192.168.1.161	192.168.1.187	OpenFlow	146	Type: OFPT_PORT_STATUS
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=17 Ack=289 Win=29056 Len=0 TSval=3990682 TSecr=3990690
192.168.1.161	192.168.1.187	OpenFlow	146	Type: OFPT_PORT_STATUS
192.168.1.187	192.168.1.161	TCP	66	6653 → 49774 [ACK] Seq=17 Ack=369 Win=29056 Len=0 TSval=3990689 TSecr=3990697
192.168.1.161	192.168.1.187	OpenFlow	146	Type: OFPT_PORT_STATUS

Figura 26 – Captura de Tráfego do estabelecimento de uma comunicação *OpenFlow*

3.4.2 Ocorrência de *ping*

Aquando da ocorrência de um *ping* entre o H1 e o H2 é gerado um ARP *request* com o objetivo do H1 conhecer o endereço MAC associado ao IP do H2. O *switch* S1, ao receber um pacote ARP *request*, uma vez que não possui qualquer informação sobre o que fazer ao pacote nas suas tabelas de *flow*, irá efetuar um pedido ao controlador para tomar uma decisão sobre o que fazer a este pacote. Desta forma é gerado um pacote *OpenFlow* do tipo *packet in* que é enviado para o controlador. Este *packet in* terá como dados o conteúdo do pacote ARP *request* como pode ser verificado na figura 27.

```
▷ Internet Protocol Version 4, Src: 192.168.1.161, Dst: 192.168.1.187
▷ Transmission Control Protocol, Src Port: 49774 (49774), Dst Port: 6653 (6653), Seq: 2746, Ack: 1767, Len: 84
◢ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 84
    Transaction ID: 0
    Buffer ID: OFP_NO_BUFFER (0xffffffff)
    Total length: 42
    Reason: OFPR_ACTION (1)
    Table ID: 0
    Cookie: 0x0000000000000000
  ▷ Match
    Pad: 0000
  ◢ Data
    ◢ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
      ▷ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      ▷ Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
        Type: ARP (0x0806)
    ◢ Address Resolution Protocol (request)
        Hardware type: Ethernet (1)
        Protocol type: IPv4 (0x0800)
        Hardware size: 6
        Protocol size: 4
        Opcode: request (1)
        Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
        Sender IP address: 10.0.0.1
        Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
        Target IP address: 10.0.0.2
```

Figura 27 - *Packet in* - ARP *Request*

Em resposta a este pedido o controlador emite um pacote do tipo *packet out* onde serão facultadas todas as informações sobre o destino que o S1 irá dar ao ARP *request* e ainda um pacote do tipo *FLOW_MOD* que sugere a criação de um *flow* apropriado ao ARP *request*.

Desta forma o S1 irá difundir o pacote para o S2 e o S3 repetindo-se o mesmo processo dado que o S2 e o S3 não têm qualquer tipo de *flow* correspondente, ao ARP *request* nas suas tabelas de *flow*

O mesmo processo irá repetir-se em todos os *switches* para qualquer outro tipo de pacotes como ARP *reply*, ICMP echo *request*, ICMP echo *reply*.

3.4.3 Manutenção do *OpenFlow Channel*

Na captura de tráfego efetuada verificou-se também a ocorrência de diversas mensagens do tipo *OFPT_ECHO_REQUEST* e *OFPT_ECHO_REPLY*. Como já mencionada estas mensagens têm a finalidade de manter a comunicação TCP ativa entre o *switch* e o controlador. As figuras 28 e 29 ilustram exemplos de pacotes *OFPT_ECHO_REPLY* e *OFPT_ECHO_REQUEST*.

```
913 48.893402   192.168.1.187   192.168.1.161   OpenFlow   74 Type: OFPT_ECHO_REQUEST
918 48.894483   192.168.1.161   192.168.1.187   OpenFlow   74 Type: OFPT_ECHO_REPLY
919 48.894489   192.168.1.187   192.168.1.161   TCP        66 6653 → 49774 [ACK] Seq=80

▷ Frame 913: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: CadmusCo_cd:c4:56 (08:00:27:cd:c4:56), Dst: CadmusCo_52:24:22 (08:00:27:52:24:22)
▷ Internet Protocol Version 4, Src: 192.168.1.187, Dst: 192.168.1.161
▷ Transmission Control Protocol, Src Port: 6653 (6653), Dst Port: 49774 (49774), Seq: 8049, Ack: 3748, Len: 8
◢ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_ECHO_REQUEST (2)
    Length: 8
    Transaction ID: 4294967282
```

Figura 28 – Captura de um pacote *OpenFlow Echo Request*

```
918 48.894483   192.168.1.161   192.168.1.187   OpenFlow   74 Type: OFPT_ECHO_REPLY
919 48.894489   192.168.1.187   192.168.1.161   TCP        66 6653 → 49774 [ACK] Seq=

▷ Frame 918: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: CadmusCo_52:24:22 (08:00:27:52:24:22), Dst: CadmusCo_cd:c4:56 (08:00:27:cd:c4:56)
▷ Internet Protocol Version 4, Src: 192.168.1.161, Dst: 192.168.1.187
▷ Transmission Control Protocol, Src Port: 49774 (49774), Dst Port: 6653 (6653), Seq: 3748, Ack: 8057, Len: 8
◢ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_ECHO_REPLY (3)
    Length: 8
    Transaction ID: 4294967282
```

Figura 29 – Captura de um pacote *OpenFlow Echo Reply*

3.5 Comunicação *Northbound*

No sentido de se desenvolverem aplicações externas ao controlador é necessário que este suporte um mecanismo de comunicação. A comunicação *northbound* do controlador *Floodlight* é definida através de um serviço REST. Um serviço REST é um serviço web que permite a disponibilização de recursos de um servidor através de pedidos simples de HTTP. A arquitetura REST define que as funções e os dados são recursos acessíveis através de URIs, isto é através de *links web*.

No controlador *Floodlight* o mecanismo de comunicação é suportado por comandos HTTP *GET*, *POST*, *PUT* e *DELETE* que devem ser emitidos pela aplicação externa com destino ao controlador sobre o porto TCP 8080. Estes comandos devem ser emitidos com o formato definido pela REST API [29]. O controlador ao receber determinado comando com determinado formato HTTP aciona a ação correspondente

ao comando solicitado respondendo na mesma ligação TCP com uma mensagem HTTP com o formato JSON ou XML.

3.5.1 Formato JSON

O JSON é uma formatação leve de troca de dados de fácil leitura para o ser humano e de fácil interpretação para máquinas. É uma formatação completamente independente da linguagem de programação a utilizar dado esta ser completamente baseada em texto.

O JSON é constituído por duas estruturas:

- Um coleção de pares nome/valor. Em várias linguagens pode ser considerado como um objeto, record, *struct*, dicionário, *hash table* e *array*. (JSONObject)
- Uma lista ordenada de valores. (JSONArray)

Em JSON, os dados são definidos segundo regras bem definidas de maneira a tornar a sua leitura normalizada:

Um objeto em JSON um conjunto desordenado de pares nome/valor. Um objeto começa com { (chave de abertura) e termina com } (chave de fecho). Cada nome é seguido por : (dois pontos) e os pares nome/valor são seguidos por , (vírgula). Tipicamente o nome é escrito entre “ (aspas) e o valor é incluído também entre “ (aspas) caso se trate de uma *string*, caso seja um número será apresentado sem aspas. [32]

Exemplo de um JSONObject:

```
{"result" : "Firewall disabled"}
```

Um *array* de objetos JSON é um conjunto de objetos JSON que tipicamente são apresentados entre [ ] (parenteses retos) contendo um ou mais objetos JSON que são iniciados e terminados pela { (chave de abertura) e terminados pela } (chave de fecho) separados por uma , (virgula).

Exemplo de um JSONArray:

```
[{"inetAddress":"/192.168.1.161:44473","connectedSince":1472312105858,"OpenFlo
wVersion":"OF_13","switchDPID":"00:00:00:00:00:00:00:02"},
{"inetAddress":"/192.168.1.161:44471","connectedSince":1472312105858,"OpenFlow
Version":"OF_13","switchDPID":"00:00:00:00:00:00:00:03"},
{"inetAddress":"/192.168.1.161:44472","connectedSince":1472312105859,"OpenFlow
Version":"OF_13","switchDPID":"00:00:00:00:00:00:00:04"},
{"inetAddress":"/192.168.1.161:44474","connectedSince":1472312105859,"OpenFlow
Version":"OF_13","switchDPID":"00:00:00:00:00:00:00:01"}]
```

Em termos programáticos, para interpretação do formato JSON foi importada uma biblioteca externa. [33]

3.5.2 Comunicação HTTP

O HTTP é um protocolo da camada de aplicação que é utilizado para transmitir todo o tipo de ficheiros pela WWW (*World Wide Web*) permitindo a transferência

ficheiros HTML (*HyperText Markup Language*), imagens, texto, etc. Uma ligação HTTP navega sobre o protocolo TCP/IP.

A comunicação HTTP funciona num modelo de comunicação entre um cliente e um servidor. Um cliente utiliza um *browser* como aplicação de cliente para efetuar pedidos HTTP a um servidor que serve de anfitrião a um servidor *web*. Por defeito um servidor possui o porto 80 à escuta de qualquer pedido HTTP.

O HTTP possui mecanismos que asseguram que o serviço esteja a funcionar corretamente. Um cliente, ao utilizar o seu *browser,* tem a possibilidade de aceder a diversos sites criando os pedidos de acesso a informação. Sempre que este pedido é respondido com sucesso o cliente receberá um resposta do servidor com um número que servirá de código de três dígitos para a interpretação do resultado da resposta.

A norma HTTP possui uma extensa lista de códigos [43] para resposta aos clientes de maneira transmitir diferentes tipos de informação. Os códigos presentes nessa lista podem ser de 5 tipos diferentes: Informativos, Sucesso, Redireccionamento, Erro de Cliente e Outros Erros.

O controlador *Floodlight* funciona como um servidor HTTP onde através do porto 8080 é disponibilizada uma interface gráfica através de um *browser*. Esta interface disponibiliza diversas funcionalidades e acesso a vários dados de forma gráfica e amigável para um administrador de sistemas. Do ponto de vista programático, o controlador *Floodlight* suporta ainda serviços web REST. Um serviço web REST consiste na disponibilização e manipulação de dados com base nos pedidos efetuados pelo cliente.

Um cliente pode efetuar pedidos a coleções inteiras de dados ou a items específicos. Existem diversos tipos de pedidos ao serviço web REST suportado pelo controlador.

A tabela 7 sintetiza a informação relativa aos vários tipos de pedidos efetuados por clientes aos servidores:

Pedido	**Função**	**A coleções (por exemplo /clientes**	**Item Especifico (por exemplo /clientes/{id}**
POST	Criação	201 (OK) Localização do item ou identificador (ID)	404 (Não encontrado), 409 (Conflito) Se já existe
GET	Leitura	200 (OK) Lista de Clientes	200 (OK) Retorna um cliente único. 404 (Não Encontrado)
PUT	Atualização/ Substituição	404 (Não Encontrado) A não ser que se queira atualizar uma coleção inteira	200 (OK), 404 (Não Encontrado)
PATCH	Atualização/ Modificação	404 (Não Encontrado) A não ser que se queira atualizar uma coleção inteira	200 (OK), 404 (Não Encontrado)
DELETE	Remoção	404 (Não Encontrado) A não ser que se queira remover uma coleção inteira	200 (OK), 404 (Não Encontrado)

Tabela 7 – Especificações dos tipos de pedido REST

A figura 30 apresenta exemplos de um pedido de recolha de informação relativa aos *switches* presentes na rede através de um pedido http do tipo GET para o URI http://192.168.1.187:8080/wm/core/controller/*switches*/json

```
46 4.413551000 192.168.1.196          192.168.1.187          HTTP          255 GET /wm/core/controller/switches/json HTTP/1.1
▶Frame 46: 255 bytes on wire (2040 bits), 255 bytes captured (2040 bits) on interface 0
▶Ethernet II, Src: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5), Dst: CadmusCo_d5:99:80 (08:00:27:d5:99:80)
▶Internet Protocol Version 4, Src: 192.168.1.196 (192.168.1.196), Dst: 192.168.1.187 (192.168.1.187)
▶Transmission Control Protocol, Src Port: 51394 (51394), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 189
▼Hypertext Transfer Protocol
 ▶GET /wm/core/controller/switches/json HTTP/1.1\r\n
  User-Agent: Java/1.8.0-ea\r\n
  Host: 192.168.1.187:8080\r\n
  Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://192.168.1.187:8080/wm/core/controller/switches/json]
```

Figura 30 – Pedido GET para o endereço http://192.168.1.187:8080/wm/core/controller/*switches*/json

Com figura 31 verifica-se uma resposta de sucesso ao pedido de GET da figura 30.

```
49 4.419575000 192.168.1.187          192.168.1.196          HTTP          71 HTTP/1.1 200 OK  (application/json)
▶Frame 49: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface 0
▶Ethernet II, Src: CadmusCo_d5:99:80 (08:00:27:d5:99:80), Dst: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
▶Internet Protocol Version 4, Src: 192.168.1.187 (192.168.1.187), Dst: 192.168.1.196 (192.168.1.196)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 51394 (51394), Seq: 1016, Ack: 190, Len: 5
▶[2 Reassembled TCP Segments (1020 bytes): #48(1015), #49(5)]
▶Hypertext Transfer Protocol
▼JavaScript Object Notation: application/json
 ▼Array
  ▼Object
   ▼Member Key: "inetAddress"
     String value: /192.168.1.161:44465
   ▶Member Key: "connectedSince"
   ▶Member Key: "openFlowVersion"
   ▼Member Key: "switchDPID"
     String value: 00:00:00:00:00:00:00:04
  ▶Object
  ▶Object
  ▶Object
```

Figura 31 – Resposta do servidor ao pedido GET de informação de *switches*

Para efetuar atualizações a campos de dados no controlador é possível efetuá-lo através do pedido PUT. A figura 32 representa o pedido de ativação do módulo *Firewall* implementado no controlador *Floodlight* através da submissão de um pedido http do tipo PUT onde é atualizado o estado do módulo para o estado *enable*.

```
154 11.12920100€ 192.168.1.196          192.168.1.187          HTTP          272 PUT /wm/firewall/module/enable/json HTTP/1.1
▶Frame 154: 272 bytes on wire (2176 bits), 272 bytes captured (2176 bits) on interface 0
▶Ethernet II, Src: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5), Dst: CadmusCo_d5:99:80 (08:00:27:d5:99:80)
▶Internet Protocol Version 4, Src: 192.168.1.196 (192.168.1.196), Dst: 192.168.1.187 (192.168.1.187)
▶Transmission Control Protocol, Src Port: 51429 (51429), Dst Port: 8080 (8080), Seq: 536, Ack: 2652, Len: 206
▼Hypertext Transfer Protocol
 ▶PUT /wm/firewall/module/enable/json HTTP/1.1\r\n
  User-Agent: Java/1.8.0-ea\r\n
  Host: 192.168.1.187:8080\r\n
  Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n
  Connection: keep-alive\r\n
 ▶Content-Length: 0\r\n
  \r\n
  [Full request URI: http://192.168.1.187:8080/wm/firewall/module/enable/json]
```

Figura 32 – Pedido de Alteração de estado do módulo *Firewall*

A figura 33 ilustra uma resposta de sucesso ao pedido efetuado na figura 32. (anterior)

```
  160 11.193900000 192.168.1.187        192.168.1.196        HTTP        594 HTTP/1.1 200 OK  (application/json)
▶Frame 160: 594 bytes on wire (4752 bits), 594 bytes captured (4752 bits) on interface 0
▶Ethernet II, Src: CadmusCo_d5:99:80 (08:00:27:d5:99:80), Dst: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
▶Internet Protocol Version 4, Src: 192.168.1.187 (192.168.1.187), Dst: 192.168.1.196 (192.168.1.196)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 51429 (51429), Seq: 2652, Ack: 742, Len: 528
▶Hypertext Transfer Protocol
▼JavaScript Object Notation: application/json
 ▼Object
  ▼Member Key: "status"
     String value: success
  ▼Member Key: "details"
     String value: firewall running
```

Figura 33 – Resposta ao pedido de alteração de estado

De maneira a efetuar submissões de novos dados no controlador como por exemplo, de novas regras no módulo *firewall*, um utilizador deve submeter um pedido do tipo POST para o endereço http://192.168.1.187:8080/wm/*firewall*/rules/json anexando o detalhe da regra ao pedido POST tal como é possível identificar na figura 34, onde é submetida uma regra que autoriza o tráfego ARP com endereço MAC de origem 00:00:00:00:00:00 e com endereço MAC destino de 00:00:00:00:00:01.

```
   90 5.012119000 192.168.1.196        192.168.1.187        HTTP        160 POST /wm/firewall/rules/json HTTP/1.1  (application/x-www-form-urlencoded)
▶Frame 90: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits) on interface 0
▶Ethernet II, Src: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5), Dst: CadmusCo_d5:99:80 (08:00:27:d5:99:80)
▶Internet Protocol Version 4, Src: 192.168.1.196 (192.168.1.196), Dst: 192.168.1.187 (192.168.1.187)
▶Transmission Control Protocol, Src Port: 51324 (51324), Dst Port: 8080 (8080), Seq: 250, Ack: 1, Len: 94
▶[2 Reassembled TCP Segments (343 bytes): #88(249), #90(94)]
▼Hypertext Transfer Protocol
 ▶POST /wm/firewall/rules/json HTTP/1.1\r\n
  User-Agent: Java/1.8.0-ea\r\n
  Host: 192.168.1.187:8080\r\n
  Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n
  Connection: keep-alive\r\n
  Content-type: application/x-www-form-urlencoded\r\n
 ▶Content-Length: 94\r\n
  \r\n
  [Full request URI: http://192.168.1.187:8080/wm/firewall/rules/json]
  [HTTP request 1/4]
  [Response in frame: 106]
▼HTML Form URL Encoded: application/x-www-form-urlencoded
 ▶Form item: "{"dl-type":"ARP","src-mac":"00:00:00:00:00:00","dst-mac":"00:00:00:00:00:01","action":"ALLOW"}" = ""
```

Figura 34 – Pedido POST de uma nova regra no módulo *Firewall*

A figura 35 ilustra uma resposta com sucesso do servidor à aceitação de uma nova regra.

```
   106 5.026682000  192.168.1.187        192.168.1.196        HTTP/XML     583 HTTP/1.1 200 OK
▶Frame 106: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface 0
▶Ethernet II, Src: CadmusCo_d5:99:80 (08:00:27:d5:99:80), Dst: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
▶Internet Protocol Version 4, Src: 192.168.1.187 (192.168.1.187), Dst: 192.168.1.196 (192.168.1.196)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 51324 (51324), Seq: 1, Ack: 344, Len: 517
▶Hypertext Transfer Protocol
▼eXtensible Markup Language
  {"status" : "Rule added", "rule-id" : "2120122370"}
```

Figura 35 – Resposta do servidor a um pedido POST bem sucedido.

Com a figura 36 verifica-se uma resposta sem sucesso do servidor à aceitação de uma nova regra dado que já existe uma regra semelhante.

```
77 3.318439000 192.168.1.187          192.168.1.196          HTTP/XML     593 HTTP/1.1 200 OK

▶Frame 77: 593 bytes on wire (4744 bits), 593 bytes captured (4744 bits) on interface 0
▼Ethernet II, Src: CadmusCo_d5:99:80 (08:00:27:d5:99:80), Dst: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
 ▶Destination: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
 ▶Source: CadmusCo_d5:99:80 (08:00:27:d5:99:80)
  Type: IP (0x0800)
▶Internet Protocol Version 4, Src: 192.168.1.187 (192.168.1.187), Dst: 192.168.1.196 (192.168.1.196)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 57525 (57525), Seq: 1, Ack: 314, Len: 527
▶Hypertext Transfer Protocol
▼eXtensible Markup Language
  {"status" : "Error! A similar firewall rule already exists."}
```

Figura 36 - Resposta do servidor a um pedido POST sem sucesso.

Relativamente aos pedidos de DELETE funcionam de forma muito semelhante aos pedidos do tipo POST. Por exemplo, para se proceder a uma remoção de uma regra um cliente deverá submeter um pedido HTTP para o endereço http://192.168.1.187:8080/wm/*firewall*/rules/json anexando o identificador da regra. A figura 37 ilustra a captura de um pacote de um pedido HTTP do tipo DELETE para a regra com *rule id* 255164164.

```
53 4.240954000 192.168.1.196          192.168.1.187          HTTP         88 DELETE /wm/firewall/rules/json HTTP/1.1  (application/x-www-form-urlencoded)
59 4.246439000 192.168.1.187          192.168.1.196          HTTP/XML     559 HTTP/1.1 200 OK

▶Frame 53: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
▶Ethernet II, Src: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5), Dst: CadmusCo_d5:99:80 (08:00:27:d5:99:80)
▶Internet Protocol Version 4, Src: 192.168.1.196 (192.168.1.196), Dst: 192.168.1.187 (192.168.1.187)
▶Transmission Control Protocol, Src Port: 51384 (51384), Dst Port: 8080 (8080), Seq: 463, Ack: 963, Len: 22
▶[2 Reassembled TCP Segments (305 bytes): #52(283), #53(22)]
▶Hypertext Transfer Protocol
▼HTML Form URL Encoded: application/x-www-form-urlencoded
 ▶Form item: "{"ruleid":"255164164"}" = ""
```

Figura 37 – Pedido DELETE a um determinado elemento do controlador *Floodlight*

A figura 38 representa uma resposta do servidor com sucesso a um pedido do tipo DELETE.

```
59 4.246439000 192.168.1.187          192.168.1.196          HTTP/XML     559 HTTP/1.1 200 OK

▶Frame 59: 559 bytes on wire (4472 bits), 559 bytes captured (4472 bits) on interface 0
▶Ethernet II, Src: CadmusCo_d5:99:80 (08:00:27:d5:99:80), Dst: HonHaiPr_cb:80:f5 (90:00:4e:cb:80:f5)
▶Internet Protocol Version 4, Src: 192.168.1.187 (192.168.1.187), Dst: 192.168.1.196 (192.168.1.196)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 51384 (51384), Seq: 963, Ack: 485, Len: 493
▶Hypertext Transfer Protocol
▼eXtensible Markup Language
  {"status" : "Rule deleted"}
```

Figura 38 – Resposta com sucesso do servidor a um pedido do tipo DELETE bem sucedido.

Na eventualidade de ser submetido um pedido de remoção de um elemento que não exista no servidor cabe ao servidor retornar uma mensagem de erro. O retorno dessa mensagem de erro pode ser efetuada através da utilização de um código de erro (por exemplo o código 404) ou através do conteúdo da mensagem HTTP. Na figura 39 surge a captura de uma resposta do servidor a um pedido HTTP do tipo DELETE para um *rule id* que não existe.

```
    45 4.874272000 127.0.0.1            127.0.0.1            HTTP        610 HTTP/1.1 200 OK  (application/json)
▶Frame 45: 610 bytes on wire (4880 bits), 610 bytes captured (4880 bits) on interface 1
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 38955 (38955), Seq: 1, Ack: 193, Len: 544
▶Hypertext Transfer Protocol
▼JavaScript Object Notation: application/json
 ▼Object
  ▼Member Key: "status"
    String value: Error! Can't delete, a rule with this ID doesn't exist.
```

Figura 39 – Resposta do controlador a um pedido DELETE sem sucesso

Verifica-se para este cenário a troca de mensagens HTTP decorreu com sucesso no entanto ocorreu uma falha que apenas pode ser detetada na camada acima, isto é, ao nível dos dados presentes na mensagem JSON existe um objeto JSON de nome status que possui um valor que indica: "*Error! Can't delete, a rule with this ID doesn't exist*.".

3.5.3 Módulos do controlador

O controlador *Floodlight* define diversas entradas na sua REST API. Esta REST API é suportada através dos diversos módulos presentes no seu código. Na página inicial do controlador é possível observar quais os módulos que estão instalados e em funcionamento.

No código de cada módulo é definido qual o URL a utilizar, qual a *class Java* a executar por determinado URL e qual o método a executar em função do tipo de mensagem HTTP recebida (*GET*, *PUT*, *POST* ou *DELETE*).

Na figura 40 estão ilustrados alguns dos módulos definidos no controlador *Floodlight*.

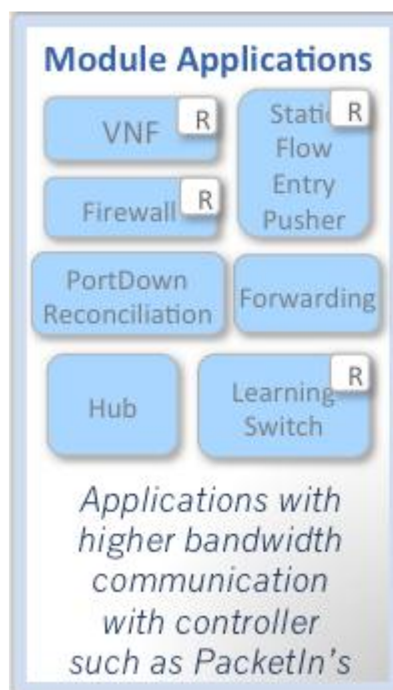

Figura 40 – Módulos *Floodlight [42]*

A API definida para cada um dos módulos está presente em [29].

Capítulo 4

Implementações Desenvolvidas

De maneira a tirar proveito das funcionalidades de um controlador SDN foram desenvolvidas duas aplicações externas ao controlador que têm como base a comunicação com o controlador utilizando alguns dos serviços que o controlador dispõe como por exemplo:

- Monitorização de débito nas portas em tempo real;
- Controlo do tráfego.

4.1 *Port Monitor*

De maneira a monitorizar uma rede SDN é possível estabelecer comunicação com o controlador, extrair a sua topologia, analisa-la e criar pontos de medição de tráfego de maneira a conseguir monitorizar o seu débito atual.

Na figura 41 encontra-se ilustrada a aplicação desenvolvida para monitorização de portos.

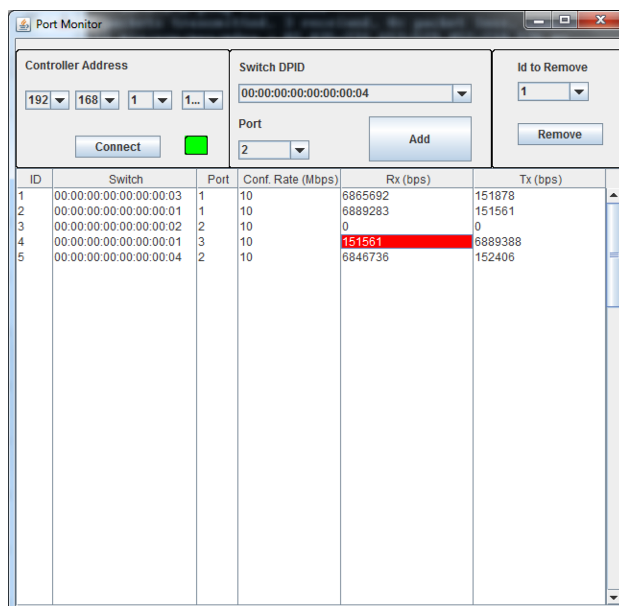

Figura 41 – Aplicação de Monitorização de portos em *switches*

Esta aplicação encontra-se desenvolvida de forma a tornar a recolha de dados bastante simples. Após o utilizador se ligar com sucesso ao controlador são habilitadas todas as funcionalidades desta aplicação. Para que tal aconteça é necessário que a aplicação externa ative o módulo *statistics* do controlador.

Após o botão *Connect* ser carregado a aplicação tenta comunicar com o controlador no sentido de fazer o levantamento da informação presente na rede, mais concretamente, quais os *switches* que existem na rede identificando-os pelo seu DPID (*Datapath ID*) e ainda quais as portas existem em cada um dos *switches*.

Tendo sido recolhida toda a topologia é possível efetuar a seleção de um conjunto *switch* e porto e carregar no botão *Add*. Ao carregar no botão *Add* será criado um ponto de monitorização para o equipamento e para o porto selecionado o que despoleta a criação de uma thread que irá medir o tráfego no porto selecionado a cada 5 segundos. Este valor foi definido como um período de tempo razoável que permite a coexistência de vários pontos de monitorização na rede sem provocar um grande aumento:

- No processamento do controlador;
- Na troca de mensagens entre controlador e a aplicação externa e
- Na troca de mensagens entre controlador e os *switches.*

O ponto de monitorização criado será ilustrado na aplicação através de uma tabela onde cada linha possui um identificador único (ID), o *switch* (*Switch*), a porta (*Port*), o débito da interface da porta (Conf. Rate) em Mbps, o débito atual na receção (RX) em bps e o débito atual na transmissão (TX) em bps.

Existe ainda um botão "*Remove*" que permite eliminar a monitorização de determinado porto com base no ID que terá sido atribuído aquando da criação do ponto de monitorização.

4.2 *Firewall* GUI

Outra aplicação externa desenvolvida consiste na interação da aplicação externa *Firewall* Gui com o módulo *Firewall* do controlador. Este módulo tem a funcionalidade de implementar um *Firewall* omnipresente na rede, ou seja, é possível impor regras em cada *switch*, em cada porta, sobre vários tipos de tráfego diferente. Estas regras podem ser impostas sobre vários parâmetros das camadas 2, 3 e 4 do modelo OSI.

Os parâmetros que definem as regras são os seguintes:

- *Switch* DPID;
- *Port;*
- *Priority;*
- *Action;*
- *Source MAC Address;*
- *Destination MAC Address;*
- *Ethertype;*
- *Source IP Address;*
- *Source IP Mask;*
- *Destination IP Address;*
- *Destination IP Mask;*
- *IP Protocol Type;*
- *IP Source Port;*
- *IP Destination Port*.

A criação de uma regra na *Firewall* por si só não significa que todos os *switches* tenham conhecimento que essa regra exista. Sendo as regras definidas no módulo, um controlador apenas comunicará determinada ação sobre determinado tipo de tráfego a um *switch*, caso este, em algum momento receba tráfego para o qual o *switch* não tenha qualquer correspondência nas suas tabelas de *flows* (*table*-miss). Tal como indicado no

capítulo 2.7.3.2, quando um *switch* deteta uma falha de correspondência sobre um pacote, o *switch* deverá remeter esse pacote para o controlador encapsulando-o dentro de um pacote do tipo *packet_in*. O controlador, recebendo o *packet_in*, tratará de enviar uma resposta ao *switch* refletindo as regras que estão implementadas no módulo *Firewall* por meio de um pacote do tipo *packet_out*. No pacote *packet_out* estará o detalhe da ação que o switch deve executar. Se ao nível do módulo controlador não exista nenhuma regra que corresponda ao pacote recebido, o controlador deve informar o *switch* para efetuar o descarte de determinado pacote.

A validade das entradas de *flow* seguirão as regras normalmente aplicadas a todas as entradas de *flow*.

Caso uma regra seja apagada do módulo *Firewall* um *switch* não se aperceberá dessa ação. Se um *switch* tiver um *flow* cuja origem esteja numa regra que atualmente já não se encontra ativa esse *flow* certamente irá expirar com o tempo dado que os *flows* possuem na sua caracterização campos de *timeout* fazem com que a validade de determinado *flow* expire com o tempo.

4.2.1 *Soft Firewall* vs *Hard Firewall*

Um dos grandes elementos diferenciadores deste módulo consiste no fato de ser possível impor regras na rede com uma precisão que pode chegar ao nível de um porto de determinado equipamento. Adicionalmente, é possível que apenas um controlador possa definir regras sobre diversas LANs (*Local Area Network*), enquanto uma *Firewall* apenas serve de mecanismo de proteção para o exterior de uma determinada LAN.

Na figura 42 verifica-se o exemplo de uma solução onde a *Firewall* física é implementada na fronteira entre a LAN e a internet.

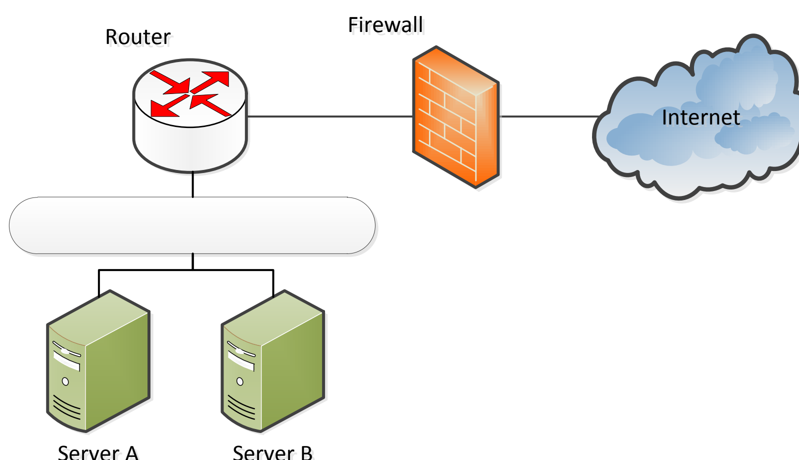

Figura 42 – Solução típica de uma rede com uma *Firewall* física.

Na figura 43 verifica-se uma solução onde a *Firewall* encontra-se distribuída por diversas redes sem a necessidade da instalação de diversos equipamentos nas instalações de clientes.

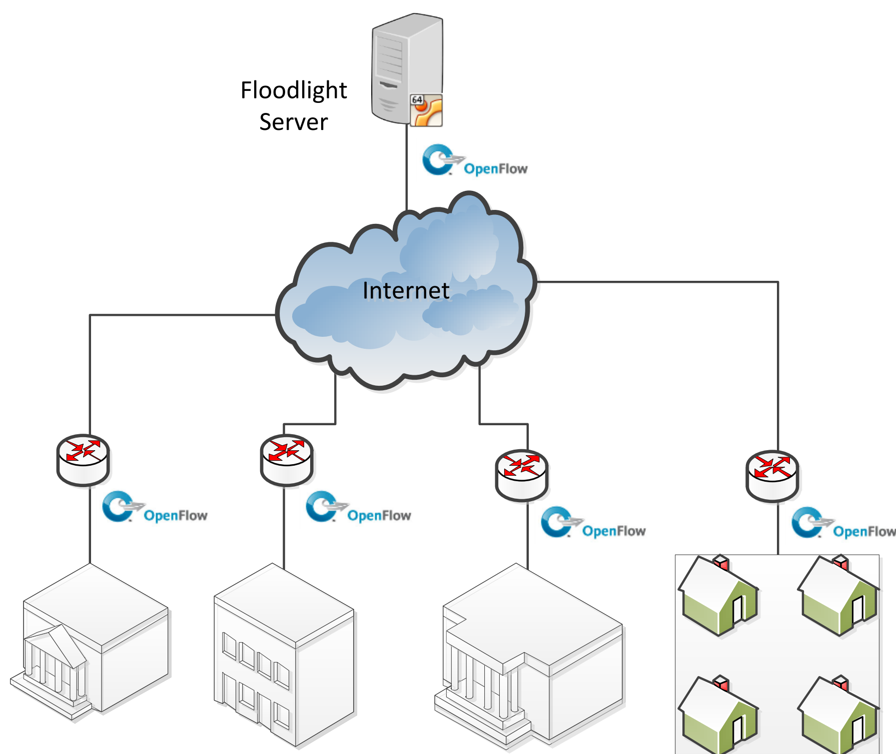

Figura 43 – Solução *OpenFlow* para um *Firewall* virtualizada.

Em termos de escalabilidade a solução apresentada na figura 43 torna-se mais vantajosa dado que não é necessário a aquisição e instalação de equipamento específico para o efeito. Um *router* com capacidade de comunicar com o controlador acaba por conseguir atuar também como *Firewall*.

4.2.2 Interação com módulo *Firewall*

Segundo a REST API do controlador *Floodlight* existem uma série de possibilidades de comandos que são possíveis de ser aplicados ao controlador no sentido de criar novos dados, extrair informações e alterar/atualizar variáveis do controlador. Para a implementação em causa tenciona-se criar um GUI que consiga interagir com o módulo *Firewall* já implementado no controlador SDN.

Para manipular os dados do módulo *Firewall* são disponibilizados vários comandos REST cuja sua descrição encontra-se disponível em [44].

De maneira a interagir com sucesso sobre o módulo *Firewall* primeiro é necessário ativar este módulo através do URI /wm/*firewall*/enable/json utilizando um pedido http do tipo PUT para que o estado do módulo seja atualizado.

De entre os comandos disponíveis destaca-se o comando REST: /wm/*firewall*/rules/json. Através deste comando e mediante a aplicação de diversos tipos de mensagem HTTP é possível efetuar todo o tipo de ações sobre o módulo como por exemplo:

- Introdução de novas regras na *Firewall* através de mensagens HTTP do tipo POST;
- Remoção de regras *Firewall* através de mensagens HTTP do tipo DELETE;
- Leitura de regras através de mensagens HTTP do tipo GET.

4.2.3 Aplicação *Firewall* Gui

De maneira a colocar o módulo do controlador à prova foi desenvolvida uma GUI que trata de facilitar a um utilizador toda a gestão das regras de uma *Firewall*.

Na figura 44 verifica-se o *layout* da aplicação externa denominada *Firewall* GUI. Nesta aplicação é possível verificar a existência de diversos painéis que tratam de melhorar a organização dos vários campos e opções disponíveis pela aplicação.

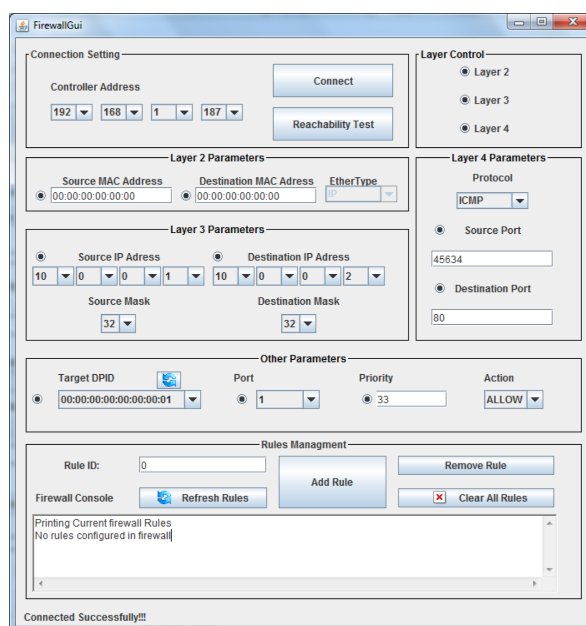

Figura 44 - Aplicação *Firewall* GUI

No painel *Connection Setting* verifica-se a possibilidade de configurar o IP do controlador bem como a possibilidade de efetuar um *Reachability test* e ainda estabelecer a ligação com o controlador. É através do botão *Connect* que é despoletado o comando REST de ativação do URI /wm/*firewall*/module/enable/json com uma mensagem http do tipo PUT. Ao estabelecer a ligação com sucesso com o controlador através do botão *Connect* todos os restantes menus desbloqueiam-se.

No painel *Layer Control* existem 3 botões que visam desbloquear/bloquear os parâmetros correspondentes a cada uma das camadas.

No painel *Layer* 2 *Parameters* encontram-se os parâmetros que descrevem a camada MAC da regra. É possível configurar qualquer endereço MAC de origem e destino no entanto o controlador apenas suporta *ethertype* do tipo ARP e IPv4. Existe a possibilidade de deixar os endereços como *wildcard*s (qualquer valor) deixando o botão correspondente ao endereço desativado. Para ter acesso a este painel é necessário o botão *Layer* 2 do painel *Layer* control estar ativo.

No painel *Layer* 3 *Parameters* encontram-se os parâmetros que descrevem a camada IP da regra. Ao nível da camada IP apenas é possível configurar os IPs origem e destino. À semelhança do painel anterior também existe a possibilidade de deixar os endereços IP de origem e destino como *wildcard*s. Para ter acesso a este painel é necessário o botão *Layer* 3 do painel *Layer Control* estar ativo. Ao ativar tal botão o campo *ethertype* é automaticamente alterado para o tipo IP.

O painel *Layer* 4 *Parameters* define os parâmetros que descrevem a camada de transporte da regra. O controlador apenas suporta 3 tipos de protocolos: TCP, UDP e ICMP. Existe também a possibilidade de configurar qualquer porto origem ou destino ou deixar qualquer um destes campos como *wildcard*s. Para ter acesso a este painel é necessário o botão *Layer* 4 do painel *Layer* control estar ativo. Ao ativar tal botão o campo *ethertype* é automaticamente alterado para o tipo IP.

Relativamente ao painel *Other Parameters* são definidos os parâmetros que não são relativos ao pacote em si. Os parâmetros definidos são:

- ***Target* DPID** – define o *switch* para o qual se quer definir a regra;
- ***Port*** – define o porto de determinado *switch* para o qual se quer definir a regra;
- ***Priority*** – define a prioridade da regra dentro do módulo. Regras com número de prioridade mais baixo possuem prioridade mais alta, isto é, uma regra com prioridade 1 é mais prioritária que uma regra com prioridade 2.
- ***Action*** – Ação a executar em caso de correspondência da regra com o pacote. O controlador apenas implementa ações de *ALLOW* (que permite a passagem de tráfego) e de DENY (que provoca o descarte do tráfego).

Neste painel existe ainda um botão com símbolo de *Refresh* que permite fazer uma atualização dos *switches* disponíveis na rede que determinado controlador gere.

Existe ainda um painel dedicado à gestão das regras existentes denominado *Rule* Management. Neste painel é possível executar ações como a leitura de todas as regras acompanhadas pelo seu *rule id* (*Refresh Rule*s), adicionar novas regras (*Add Rule*), remover apenas uma regra (*Remove Rule*) ou ainda remover todas as regras (*Clear All Rule*s). Para executar estas ações foi utilizado exclusivamente o URI /wm/firewall/rules/json efetuando as diversas ações de leitura, criação e remoção através de pedidos http do tipo GET, POST e DELETE respetivamente. é

Relativamente à adição de regras, ao carregar no botão *Add Rule* a aplicação comunicará com o controlador por forma a criar uma regra na máquina que possua todos os parâmetros que estão configurados na aplicação. Por cada regra adicionada o controlador responde com o estado da ação (se a regra foi ou não introduzida com sucesso. Caso a regra tenha sido introduzida com sucesso será atribuída um número denominado *Rule* ID que identificará de forma única a regra. Aquando de uma remoção de uma regra deverá ser utilizado o *Rule* ID no campo de texto *Rule* ID por forma a *remove*r a regra pretendida.

No fundo da aplicação existe uma barra de estado que irá comunicado ao utilizador diferentes estados relativamente ao sucesso ou falha de determinadas ações.

4.2.4 Comunicação *southbound* das regras

A comunicação *southbound* das regras será refletida através de mensagens *OpenFlow*. Tendo o módulo *Firewall* ativado por defeito todo o tráfego *unicast* dos equipamentos geridos pelo controlador é descartado pelo que é necessário que existam regras aplicadas à rede para que a mesma possa funcionar.

De maneira a exemplificar a comunicação *southbound* do módulo *firewall* foi desenvolvida a topologia ilustrada na figura 45:

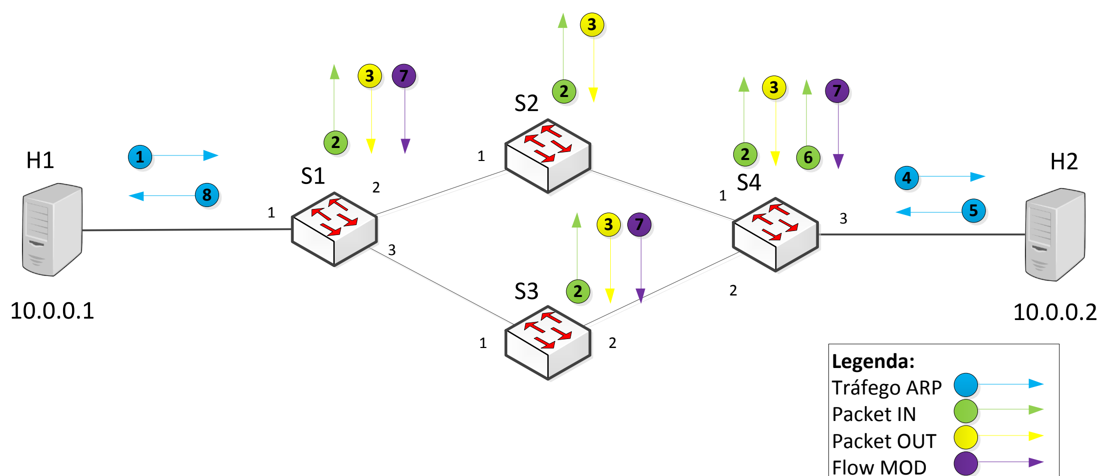

Figura 45 – Processamento de uma mensagem ARP numa rede *OpenFlow*

Com as seguintes regras implementadas:

- *Rule id: -897945047 , Switch: 00:00:00:00:00:00:00:02, Source Mac Address: 00:00:00:00:00:01, Ethertype: 0x0806, Priority: 1, Action: DROP*
- *Rule id: 85496119 , Ethertype: 0x0800, Priority: 10, Action: ALLOW*
- *Rule id: -1731541199, Ethertype: 0x0806, Priority: 10, Action: ALLOW*

As regras indicam que o *switch* S2 não permite qualquer tráfego ARP com endereço de MAC de origem 00:00:00:00:00:01, enquanto as outras 2 regras dizem respeito à aceitação total por parte de todos os *switches* do tráfego ARP e do tráfego IPv4. Tendo a regra de rule id -897945047 prioridade 1, torna-se a regra mais prioritária pelo que é correto afirmar que todo o tráfego ARP no *switch* s2 será descartado.

Na figura 45 o *Host* h1 pretende descobrir qual o endereço de MAC atribuído ao IP 10.0.0.2. Para tal o *Host* h1 difunde um pacote do tipo ARP *request* para o *switch* S1. Desde a emissão do pacote ARP *request* até à receção do pacote ARP *reply* por parte do h1 procedem-se diversas trocas de mensagens entre *Hosts*, *switches* e controlador. Fazendo uso das numerações presentes na figura 45 descrevem-se as seguintes trocas de mensagem:

1. *Host* h1 difunde um pacote ARP *Request* para o *Switch* S1;
2. O pacote ARP *Request* é difundido por todo o domínio de broadcast, isto é por todos os *switches* sendo que todos os *switches* ao receberem tal pacote emitem um *packet in* para o controlador;
3. O controlador processa o pacote *packet in* analisando o seu conteúdo. Analisando a topologia da rede, o controlador, verifica que ainda não conhece o endereço MAC para o IP 10.0.0.2 pelo que emite um *packet out* para todos os *switches* com informação para procederem à difusão do ARP *request* por todas as suas portas;

4. O *Host* h2 recebe o pacote de ARP *Request* com sucesso por difusão do pacote pelo *switch* S4;
5. O *Host* h2 emite um pacote ARP *Reply* para o S4;
6. O S4 emite um pacote *packet in* em direção ao controlador;
7. O controlador processa o *packet in* e com base nas regras que possui no módulo *firewall* emite pacotes do tipo *FLOW* MOD apenas para os *switches* S1, S3 e S4;
8. O ARP *reply* é recebido com sucesso.

Aquando da emissão do pacote *flow mod* o controlador recorre ao conhecimento que tem da topologia e às regras que possui na *firewall* para difundir os *flows* necessários apenas para os *switches* que fazem parte do caminho que algoritmo de *Dijkstra* calculou.

4.2.5 Testes de performance

De maneira a avaliar o impacto da utilização do módulo *Firewall* numa rede SDN foram efetuados 2 testes sobre a topologia descrita na figura 19. Os testes efetuados baseiam-se na execução de 1000 *pings* entre o h1 e o h2 onde numa primeira abordagem teremos o h1 a efetuar *pings* sobre o h2 sem qualquer tipo de *Firewall* ativa enquanto numa segunda abordagem teremos o h1 a efetuar *pings* sobre o h2 com o módulo *Firewall* ativo com regras apropriadas para que os pings sucedam com sucesso. A cada *hop* foi imposto um *delay* de 10ms.

Analisando as figuras 46 e 47 é possível observar os resultados de ambos os cenários.

```
--- 10.0.0.2 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 999806ms
rtt min/avg/max/mdev = 80.904/82.643/179.262/3.235 ms
```

Figura 46 – Execução de 1000 *pings* entre 2 *Hosts* sem *Firewall* ativa

```
--- 10.0.0.2 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 999774ms
rtt min/avg/max/mdev = 80.987/82.471/177.458/3.088 ms
```

Figura 47 – Execução de 1000 *pings* entre 2 *Hosts* com *Firewall* ativa

Comparando os resultados de ambos os testes não são evidentes grandes diferenças em termos de performance. Em ambos os cenários, o 1º *ping* demorou cerca de 178ms devido à troca de mensagens *OpenFlow* entre *switch*, já os restantes *pings* mantiveram o seu valor médio de 82.5ms.

4.3 Otimização ao Módulo *Firewall*

Aquando do desenvolvimento da *Firewall* GUI constatou-se que o sistema de aceitação de regras na *Firewall* não estaria a ser o mais correto.

4.3.1 Formulação do problema

Analisando o código presente na classe *net.Floodlightcontroller.Firewall .Firewall RulesResource.Java* foi possível desenvolver o fluxograma apresentado na figura 48.

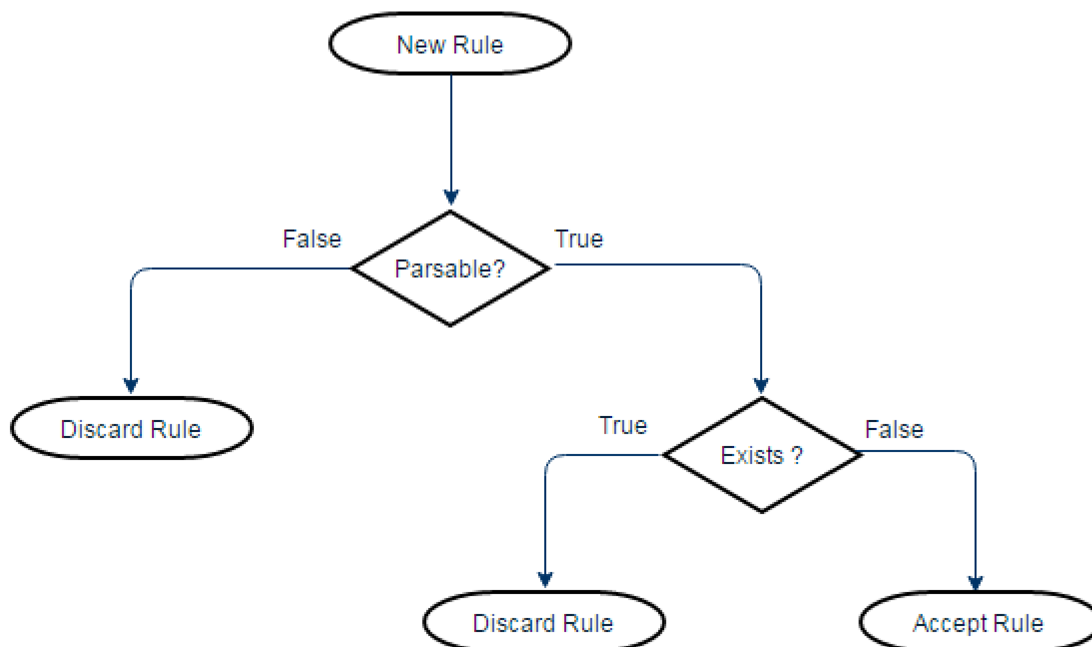

Figura 48 – Fluxograma da aceitação de uma regra na *Firewall* do controlador *Floodlight*

Nesta figura verifica-se que aquando da receção de uma regra são apenas efetuadas 2 verificações:

- É verificada a integridade dos dados, isto é, se os dados estão em formato JSON e se estes dados definem alguma regra;
- Tendo recebido uma regra bem formatada é efetuada uma comparação da nova regra com todas as regras presentes no *Firewall* . Caso esta regra seja diferente de todas as restantes a regra deverá ser aceite. Caso contrário os dados deverão ser descartados.

Analisando o algoritmo de aceitação de regras é possível identificar que o mesmo não é o mais robusto e que este carece de um aumento de critérios aquando da aceitação. Com o atual algoritmo é possível aceitar regras que tanto aceitem o tráfego como rejeitem o tráfego, isto é, é possível aceitar regras com precisamente todos os parâmetros iguais (prioridade incluída) à exceção do parâmetro *Actions*. Adicionalmente é possível também aceitar regras que se sobreponham umas às outras tornando a lista de regras da *Firewall* algo redundante, contraditória e complexa.

4.3.2 Proposta de alteração

Tendo em conta que o projeto *Floodlight* é um projeto de fonte aberta foram trocados diversos *e-mails* com a equipa de desenvolvimento da *Floodlight* de forma a reportar o problema identificado. Ficou acordado que no âmbito desta dissertação seria criado um novo algoritmo de aceitação de regras que fosse mais criterioso de forma a

tornar o módulo da *Firewall* mais robusto e preciso foi então proposto o seguinte fluxograma apresentado na figura 49:

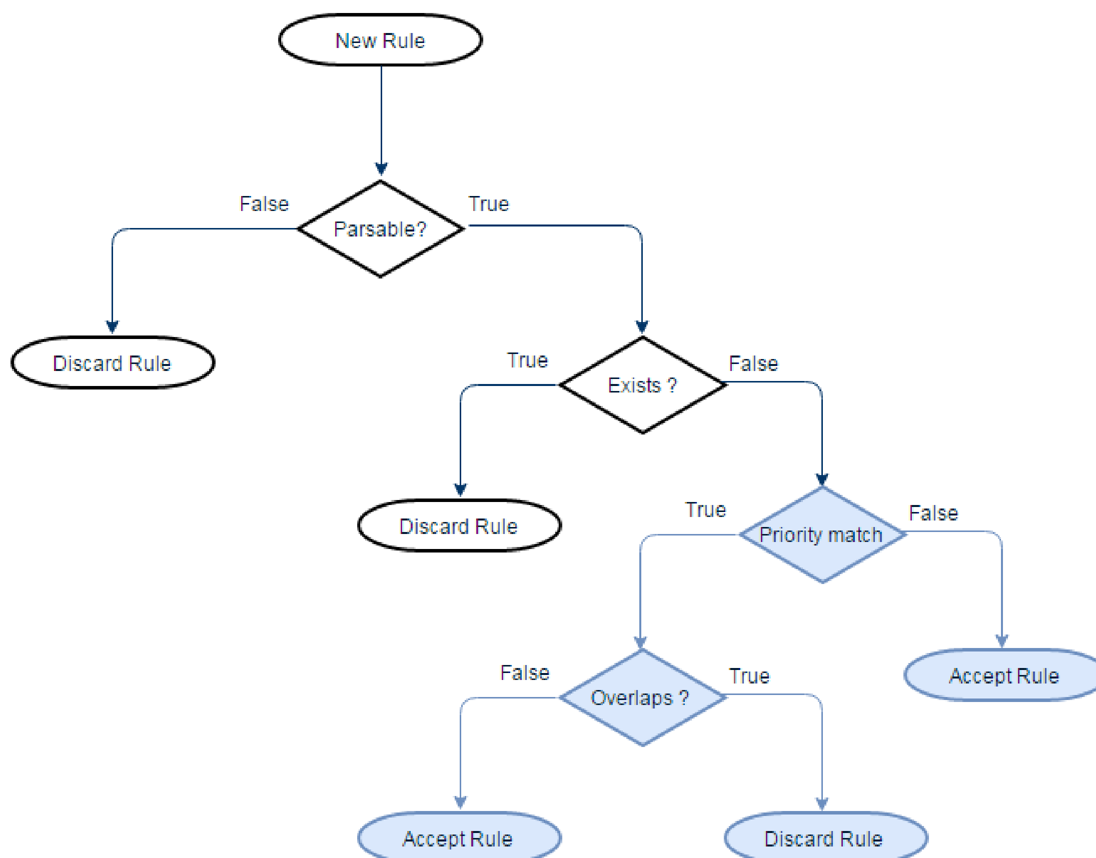

Figura 49 – Novo fluxograma de aceitação de regras na *Firewall Floodlight*

Com o fluxograma ilustrado verifica-se um aumento nos critérios definidos para aceitação de regras.

Foi efetuada uma alteração na decisão *"Exists ?"*. Inicialmente neste ponto a nova regra seria comparada a todas as regras sobre todos os parâmetros que definem uma regra. Foi efetuada uma alteração passando a não considerar o campo *Action* como parâmetro a comparar. Se este campo estiver incluído no algoritmo de verificação de existência da nova regra na lista de regras atualmente configuradas corre-se o risco de se aceitarem duas regras iguais com campos *Action* diferentes, por exemplo, aceitar todo o tráfego do *switch* A e rejeitar todo o tráfego do *switch* B. Neste exemplo as duas regras não poderão coexistir.

Após verificar a existência da regra ou não dentro da lista de regras a nova regra é sujeita a mais uma série de verificações que têm em vista a redução do número de regras redundantes e conflituosas.

Neste sentido foi desenvolvido uma função denominada *checkRuleOverlap* que consiste na verificação de sobreposições de parâmetros como critério para a aceitação de uma nova regra. Caso uma nova regra tenha a capacidade de se sobrepor a uma regra já configurada na *Firewall* esta regra deve ser rejeitada. Uma nova regra não pode sobrepor-se a uma regra já configurada com a mesma prioridade.

O algoritmo desenvolvido consiste na comparação dos diversos parâmetros da nova regra com todas as regras atuais. A cada regra atualmente configurada é verificado de imediato o campo de prioridade, caso a prioridade da nova regra não seja igual à prioridade da regra atual a nova regra é aceite.

Para verificação de sobreposição foram criadas duas variáveis do tipo *Interger* denominadas de *overlap* e de *sameField*. Para estas variáveis tenciona-se apenas utilizar 10bit em cada variável por forma a guardar o resultado das várias comparações a efetuar entre a nova regra e as regras já configuradas.

DPID	IN_*PORT*	DL_SRC	DL_DST	DL_TYPE	NW_SRC	NW_DST	NW_PROTO	TP_SRC	TP_DST
1	2	4	8	16	32	64	128	256	512

Tabela 8 – Parâmetros presentes nas regras a comparar e valor de bit atribuído nas variáveis *overlap* e *sameField*

A variável *overlap* guarda o resultado do XOR dos valores que indicam a presença de wild-card da nova regra e da regra atual. O primeiro bit corresponde à comparação dos campos new*Rule*.*Any*_DPID XOR current*Rule*.*Any*_DPID, os restantes bits possuem o valor correspondente ao XOR dos campos da ordem descrita na tabela8.

Já a variável *sameField* guarda o resultado da comparação entre os parâmetros da nova regra e da regra atual. O primeiro bit corresponde à comparação dos campos new*Rule*.DPID == current*Rule*.DPID, os restantes bits possuem o valor correspondente à comparação com base na ordem descrita na tabela 7.

Ao terminar as comparações dos parâmetros de cada camada é verificada a condição de sobreposição. Apenas se procede o algoritmo caso não se esteja ainda na condição de sobreposição. Se após terminar a comparação dos parâmetros da camada de transporte não se verificar qualquer sobreposição significa que a nova regra não se sobrepõe a qualquer uma das regras atualmente configuradas na *Firewall* e como tal a nova regra deverá ser aceite na *Firewall*.

A condição de sobreposição segue a seguinte expressão:

$$(overlap > 0)\ \&\&\ ((overlap \mid sameField) == NUMBER)$$

De maneira a compreender melhor a expressão será necessário considerar como primeira condição que *overlap* tem de ser superior a 0, isto é tem de existir pelo menos um parâmetro que se sobreponha a outro parâmetro. Como segunda condição segue uma expressão que reflete o operador OR bit a bit entre os parâmetros *overlap* e *sameField* seguido de uma comparação com o valor NUMBER.

O valor NUMBER corresponde ao valor de cada camada com todos os bits a 1, isto é, a cada camada o valor NUMBER muda consoante o número de variáveis que já foram comparadas, por exemplo, ao terminar a camada física foram comparados dois parâmetros (DPID e IN_*PORT*) o que totaliza 2 bits a 1 logo NUMBER = 3. Ao final da camada *datalink* NUMBER = 31, para a camada IP NUMBER = 127 e para camada de transporte NUMBER = 1027.

Dada a complexidade da expressão $(overlap \mid sameField) == NUMBER$ esta será explicada recorrendo a alguns exemplos que auxiliam a sua compreensão. Os

seguintes exemplos apenas compreenderão a camada física para expor com simplicidade a expressão dado que a adição de mais camadas resultará nas mesmas conclusões.

Na tabela 9 verifica-se um exemplo de sobreposição. A regra A está a sobrepor-se à regra B. Ao nível de variáveis o *overlap* é igual 2 e o *sameField* é igual 1, o OR bit a bit corresponde a 3 e o *overlap* é superior a 0, logo existe sobreposição.

Regras\	DPID	IN_*PORT*
A	00:00:00:00:00:00:00:01	*Any*
B	00:00:00:00:00:00:00:01	1

Tabela 9 – Cenário de sobreposição de regras.

Na tabela 10 existe uma sobreposição evidente. O *overlap* é igual a 3 enquanto o *sameField* é igual a 0, o *overlap* é também superior a 0 logo temos sobreposição.

Regras	DPID	IN_*PORT*
A	*Any*	*Any*
B	00:00:00:00:00:00:00:01	1

Tabela 10 – Sobreposição em múltiplos parâmetros.

Na tabela 11 o *overlap* é igual a 2 mas o *sameField* é igual a 0 logo não existe sobreposição dado que a regra A define uma nova regra para o DPID 00:00:00:00:00:00:00:02 enquanto a regra B define uma nova regra para o DPID 00:00:00:00:00:00:00:01

Regras	DPID	IN_*PORT*
A	00:00:00:00:00:00:00:02	*Any*
B	00:00:00:00:00:00:00:01	1

Tabela 11 – Regras não entram em sobreposição

Com a tabela 12 colocamos o algoritmo à prova num cenário de sobreposição múltipla. A Regra A sobrepõe-se à regra B pelo DPID enquanto a regra B sobrepõe-se à regra A pelo campo IN_*PORT*. A variável *overlap* será igual a 3 logo existirá uma sobreposição.

Regras	DPID	IN_*PORT*
A	*Any*	1
B	00:00:00:00:00:00:00:01	*Any*

Tabela 12 – Sobreposição de regras mutua.

Na tabela 13 é possível observar um conjunto de regras totalmente diferentes. Logo não existirá certamente sobreposição. O campo *sameField* é igual a 0 tal como o campo *overlap*.

Regras	DPID	IN_*PORT*
A	00:00:00:00:00:00:00:02	2
B	00:00:00:00:00:00:00:01	1

Tabela 13 – Regras Diferentes

Ao nível do algoritmo existe um parâmetro que não entra em consideração, o campo *Action*. Uma vez que o campo *Action* apenas assume 2 valores (*ALLOW*, DENY) não faria sentido que este entrasse no algoritmo pois este campo não interfere na

caracterização de um par de regras sobre o facto de estas se sobreporem ou não uma face a outra. Ao não comparar este campo também se exclui a hipótese de uma regra não entrar em conflito com outra regra igual.

4.3.3 Aceitação da proposta

O código fonte do controlador encontra-se disponibilizado na plataforma GitHub em [40]. A plataforma GitHub [39] consiste num portal onde é possível programadores publicarem o código fonte de diversas aplicações com o intuito de promover a partilha e desenvolvimento pela comunidade científica.

Através deste portal foi efetuada a submissão do contributo através do *pull request* nº 715 [41] com a designação de *Firewall Optimization*.

Capítulo 5

Conclusões

O estudo apresentado por esta dissertação ilustra que o SDN é algo a ter em conta para os próximos tempos e provavelmente será parte integrante de diversas redes num futuro próximo. Tendo em conta o poder programático que a arquitetura SDN faculta, é possível afirmar que cada vez mais existirão automatismos nas redes o que possibilitará uma expansão mais rápida das redes e uma redução dos vários custos operacionais que as implementações de novos serviços acarretam. É necessário ter em conta que as redes SDN são hoje já uma realidade mas no entanto a tecnologia ainda se encontra numa fase de maturação tendo em conta que já existem diversos organismos internacionais dedicados à definição de normas e protocolos.

De maneira a simplificar a definição das redes SDN foi criado um ambiente de testes em ambiente virtual onde foi possível verificar com elevado detalhe a forma de comunicação das diferentes camadas da arquitetura SDN. Foi possível virtualizar uma rede com uma dada topologia através do software *Mininet* colocando os respetivos elementos de rede em comunicação com o controlador *Floodlight*. Tendo o ambiente de testes criado foi possível efetuar capturas de pacotes através do *wireshark* e analisar pormenorizadamente a troca de mensagens *OpenFlow* de forma a constatar o correto funcionamento do protocolo entre ambas as máquinas virtuais.

Ao nível da comunicação *northbound* do controlador foram desenvolvidas diversas ferramentas de maneira a permitir a correta comunicação de uma aplicação *Java* com a plataforma do controlador *Floodlight* passando os respetivos argumentos na formatação JSON. Após construir todas as ferramentas necessárias para permitir uma boa comunicação com o controlador foram ainda desenvolvidas duas aplicações também em *Java* no sentido de ilustrar as capacidades de um controlador SDN. Inicialmente terá sido desenvolvida uma aplicação que visa a monitorização de tráfego constante nos portos *ethernet* na rede. Tendo essa aplicação implementada e em funcionamento foi também desenvolvida uma aplicação que visa interagir com a plataforma *Floodlight* ao nível do seu módulo de *Firewall* em ambiente gráfico.

No decorrer do desenvolvimento da aplicação *Firewall* foi possível identificar uma situação que apresentava não ser a mais correta no que toca às boas práticas de funcionamento de uma *Firewall*. O modelo de aceitação de regras na *Firewall* possibilitava a existência de regras que se sobreporiam umas às outras e que se contradiziam umas às outras tornando a experiência do ponto de vista de um utilizador pouco fidedigno e de análise complexa. Foi então desenvolvida e apresentada uma proposta de alteração deste modelo à equipa de desenvolvimento da *Floodlight* no sentido de otimizar o modelo de aceitação de regras na *Firewall*.

As redes SDN revelam ser a chave que irá trazer um novo fôlego às redes de telecomunicações tradicionais. A partir do momento em que o acesso programático é desbloqueado as possibilidades são infinitas e em diversos níveis como por exemplo no desenvolvimento de equipamento, na implementação de serviços e da automatização de configurações.

5.1 Trabalho Futuro

No que respeita ao trabalho futuro existem diversas possibilidades. Através da biblioteca *Java* que foi desenvolvida para comunicação, em específico, com o controlador *Floodlight* é possível desenvolver diversas funcionalidades extraindo exclusivamente dados a partir do controlador.

Existe a possibilidade de serem desenvolvidos novos ambientes de simulação com base nas máquinas virtuais construídas, por exemplo, para estudos na área da segurança é possível desenvolver cenários de ataques de DDOS (*Distributed Denial of Service*) com vários *Hosts* no sentido de efetuar testes a mecanismos de segurança em redes SDN ou até mesmo em redes tradicionais uma vez que a plataforma *Mininet* pode funcionar de forma independente.

Ao nível da plataforma *Floodlight,* sendo este um sistema de código de fonte aberta, existe a possibilidade de se desenvolver novo código no sentido de disponibilizar novos módulos como por exemplo a aplicação de medidas de segurança de forma dinâmica ou até mesmo desenvolvimento de um sistema de alarmísticas face a determinados eventos que ocorram na rede monitorizada pelo controlador.

Referências

[1] – Jianying Luo and Justin Pettit and Martin Casado and John Lockwood and Nick McKeown, Prototyping Fast, Simple, Secure Switches for Ethane, Hot Interconnects, Stanford, Agosto 2007

[2] – Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown and Scott Shenker, Ethane: Taking Control of the Enterprise, Japão, Agosto 2007

[3] – Vitor Almeida, Redes de Internet, Instituto Superior de Engenharia de Lisboa, Setembro 2011.

[4] – Tom Nolle, SDN, NFV, and "Higher Layers", Julho 2014

[5] – Open Network Foundation: What and Why ?: https://www.opennetworking.org/images/stories/downloads/about/onf-what-why.pdf (Dezembro 2015)

[6] – Arjun Singh, Joon Ong, Google , Inc.: Jupiter Rising: A Decade of Clos topologies and Centralized Control in Google 's Datacenter Network (Agosto 2015)

[7] – Nathan Farrington, Erik Rubow, and Amin Vahdat: Data Center Switch Architecture in the Age of Merchant Silicon (Agosto 2009)

[8] – Yuval Bachar, Adam Simpkins: Introducing wedge and FBOSS the next steps toward a disagregated network, Junho 2014

[9] – Yuval Bachar: Introducing "6-pack": the first open hardware modular switch, Fevereiro 2015

[10] – Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky and Steve Uhlig, "Software-Defined Networking:A Comprehensive Survey", Proceedings of the IEEE, Volume 103, Issue 1, Jan.2015.

[12] – http://archive.OpenFlow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT , Website acedido a Novembro 2015

[13] – Steve Roberts, "Turning the Raspeberry Pi into an OpenFlow Switch, Market Footprint, Setembro 2013.

[14] – http://Mininet.org/ , Website acedido a Março 2015.

[15] –https://www.opennetworking.org/sdn-resources/sdn-definition Website acedido em Dezembro 2015

[16] – http://www.frank-durr.de/?p=68 Website acedido em Janeiro 2015

[17] – Michael Jarschel, Michael Jarschel, Tobias Hossfeld, Interfaces, attributes, and use cases: A compass for SDN, IEEE Communications Magazine, Junho 2014.

[18] – http://networkheresy.com/2011/10/27/origins-and-evolution-of-OpenFlowsdn/ Website acedido em Novembro 2015

[19] – Open Flow Switch Specification 1.5.1, May 2015, Open Networking Foundation.

[20] – Can OpenFlow Scale ? Bob Lynch, Junho 2013, SDX Central.

[21] – OpenFlow 1.5.1 Jung Byeonghwa, PioLink, Inc. 2015

[22] – David Jorm, SDN and Security, Abril 2015

[23] – Jim duffy, Cisco on why OpenFlow alone doesn't cut it, Network World, Junho 2015.

[24] – Bobbie Johnson, Is OpenFlow an answer looking for a problem ?, Junho 2012.

[25] – https://saas.hpe.com/marketplace/sdn#/Home/Show Website acedido em Novembro 2015

[26] – Sridhar Rao, Comparison Of Open Source SDN Controllers, The New Stack, Março 2015.

[27] – http://openvswitch.org/ Website acedido em Dezembro 2015

[28] – https://Floodlight.atlassian.net/wiki/display/Floodlightcontroller/Module+Descriptions+and+Javadoc, Website acedido em Junho 2016

[29] – https://Floodlight.atlassian.net/wiki/display/Floodlightcontroller/Floodlight+REST+API , Website acedido em Junho 2016

[30] – Mario Cavalli, "Mission Critical ClusterInfrastructure, HyperTransport Technology Consortium, Ethernet Technology Summit 2012.

[31] - http://www.projectFloodlight.org/download/ Website acedido em Abril 2016

[32] – www.json.org ,Website acedido em Junho 2016

[33] – https://github.com/stleary/JSON-Java, Website acedido em Junho 2016

[34] – Jon Matias, Jokin Garay, Nerea Toledo, Toward an SDN-enabled NFV architecture, IEEE Communications Magazine, Abril 2015

[35] – https://www.sdxcentral.com/resources/events/ Website acedido em Novembro 2015

[36] – http://events.linuxfoundation.org/events/linuxcon-europe/program/schedule Website acedido em Novembro 2015

[37] – https://en.wikipedia.org/wiki/OSI_model Website acedido em Outubro 2015

[38] – Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, "Towards Secure and dependable software-defined networks", HotSDN'13, China, Agosto 2013.

[39]– https://github.com/ , Website acedido em Junho 2016

[40] – https://github.com/Floodlight/Floodlight, Website acedido em Junho 2016

[41] – https://github.com/Floodlight/Floodlight/pull/715, Website acedido em Junho 2016

[42] – https://Floodlight.atlassian.net/wiki/display/Floodlightcontroller/Architecture , Website acedido em Junho 2016

[43] – https://pt.wikipedia.org/wiki/Lista_de_c%C3%B3digos_de_estado_HTTP, Website acedido em Julho 2016

[44] – https://Floodlight.atlassian.net/wiki/display/Floodlightcontroller/Firewall+REST+API , Website acedido em Junho 2016

[45] – https://Floodlight.atlassian.net/wiki/display/Floodlightcontroller/Floodlight+Services, Website acedido em Junho 2016

[46] – Melissa Yan, Dijkstra's Algorithm, Massachusetts Institute of Technology, Janeiro 2014

[47] – R. Sherwood "Tutorial: White Box/Bare Metal Switches", Open Networking User Group meeting, New York, May 2014

[48] –T. Berners – Lee, W3C/MIT, RFC 3986, Janeiro 2005.

Anexos

A. Diferenças de capacidades entre Versões *OpenFlow*

OpenFlow Version	Match fields	Statistics	# Matches		# Instructions		# Actions		# Ports	
			Req	Opt	Req	Opt	Req	Opt	Req	Opt
v 1.0	Ingress Port Ethernet: src, dst, type, VLAN IPv4: src, dst, proto, ToS TCP/UDP: src port, dst port	Per table statistics Per flow statistics Per port statistics Per queue statistics	18	2	1	0	2	11	6	2
v 1.1	Metadata, SCTP, VLAN tagging MPLS: label, traffic class	Group statistics Action bucket statistics	23	2	0	0	3	28	5	3
v 1.2	OpenFlow Extensible Match (OXM) IPv6: src, dst, flow label, ICMPv6		14	18	2	3	2	49	5	3
v 1.3	PBB, IPv6 Extension Headers	Per-flow meter Per-flow meter band	14	26	2	4	2	56	5	3
v 1.4	—	— Optical port properties	**14**	**27**	**2**	**4**	**2**	**57**	**5**	**3**

Figura 50 – Capacidades das diferentes versões *OpenFlow* [10]

B. Código da alteração ao módulo *Firewall*

Ao nível do contributo efetuado no desenvolvimento e correção do módulo *Firewall* foi adicionado um método novo, `checkRuleOverlap` e ainda foi efetuada uma alteração no método store de maneira a refletir a proposta de alteração.

O seguinte código ilustra a alteração que foi efetuada sobre o método store:

```
@Post
public String store(String fmJson) {
      IFirewallService firewall =
                  (IFirewallService)getContext().getAttributes().
                  get(IFirewallService.class.getCanonicalName());

      FirewallRule rule = jsonToFirewallRule(fmJson);
      if (rule == null) {
            return "{\"status\" : \"Error! Could not parse firewall
rule, see log for details.\"}";
      }
      String status = null;
      if (checkRuleExists(rule, firewall.getRules())) {
            status = "Error! A similar firewall rule already exists.";
            log.error(status);
            return ("{\"status\" : \"" + status + "\"}");
      } else {
            // add rule to firewall
            String res = checkRuleOverlap(rule, firewall.getRules());
            if(res != null){ // isOverlapable

                  status = "Rule Not added";
                  log.error(res);
                  return ("{\"status\" : \"" + status + "\"}");
            }
```

```
            firewall.addRule(rule);
            status = "Rule added";
            return ("{\"status\" : \"" + status + "\", \"rule-id\" :
\""+ Integer.toString(rule.ruleid) + "\"}");

      }
}
```

O seguinte código ilustra o código do novo método que verifica a sobreposição de regras:

```
public static final int DPID_BIT    = 1;
public static final int IN_PORT_BIT = 2;
public static final int DL_SRC_BIT  = 4;
public static final int DL_DST_BIT  = 8;
public static final int DL_TYPE_BIT= 16;
public static final int NW_SRC_BIT  = 32;
public static final int NW_DST_BIT  = 64;
public static final int NW_PROTO_BIT= 128;
public static final int TP_SRC_BIT  = 256;
public static final int TP_DST_BIT  = 512;
public static final String NEW_RULE_OVERLAPS = "WARNING: This rule
overlapes another firewall rule with rule id: ";
public static final String NEW_RULE_OVERLAPED = "WARNING: The rule is
overlaped by another firewall rule with rule id: ";

/**
 * Checks for Rule Overlaping in following conditions
 * New rule having priority equal to current rules
 * New rule having having equal parameters and wildcards
 * @param rule - the new rule
 * @param rules - rules list
 * @return error a String error message. Null if no overlap event is
found
 */
public static String checkRuleOverlap(FirewallRule rule,
List<FirewallRule> rules) {
      Iterator<FirewallRule> iter = rules.iterator();
      // Loops throught all Rules
      while (iter.hasNext()) {

            FirewallRule r = iter.next();
            // Priority check
            if(rule.priority == r.priority){
                  int overlap = 0;
                  // if true , new rule overlapes, false new rule is
overlaped
                  boolean whoOverlapes = false;
                  int sameField = 0;

                  // Check Switch Overlap
                  if(rule.any_dpid ^ r.any_dpid){
                        overlap += DPID_BIT;
                        whoOverlapes = (rule.any_dpid && !whoOverlapes)
? true : false;
                  }
                  if(rule.any_in_port ^ r.any_in_port){
                        overlap += IN_PORT_BIT;
                        whoOverlapes = (rule.any_in_port &&
!whoOverlapes) ? true : false;
                  }
```

```
                    if(rule.dpid.equals(r.dpid))
                          sameField += DPID_BIT;
                    if(rule.in_port.equals(r.in_port))
                          sameField += IN_PORT_BIT;
                    if((overlap | sameField) == 3 && overlap > 0)
                          return ((whoOverlapes) ? NEW_RULE_OVERLAPS :
NEW_RULE_OVERLAPED) + r.ruleid;

                    // Check Layer 2 Overlap
                    if(rule.any_dl_src ^ r.any_dl_src){
                          overlap += DL_SRC_BIT;
                          whoOverlapes = (rule.any_dl_src &&
!whoOverlapes) ? true : false;
                    }
                    if(rule.any_dl_dst ^ r.any_dl_dst){
                          overlap += DL_DST_BIT;
                          whoOverlapes = (rule.any_dl_dst &&
!whoOverlapes) ? true : false;
                    }
                    if(rule.any_dl_type ^ r.any_dl_type){
                          overlap += DL_TYPE_BIT;
                          whoOverlapes = (rule.any_dl_type &&
!whoOverlapes) ? true : false;
                    }
                    if(rule.dl_src.equals(r.dl_src))
                          sameField += DL_SRC_BIT;
                    if(rule.dl_dst.equals(r.dl_src))
                          sameField += DL_DST_BIT;
                    if(rule.dl_type.equals(r.dl_type))
                          sameField += DL_TYPE_BIT;
                    if((overlap | sameField) == 31 && overlap > 0)
                          return ((whoOverlapes) ? NEW_RULE_OVERLAPS :
NEW_RULE_OVERLAPED) + r.ruleid;

                    // Check Layer 3 Overlap
                    if(rule.any_nw_src ^ r.any_nw_src){
                          overlap += NW_SRC_BIT;
                          whoOverlapes = (rule.any_nw_src &&
!whoOverlapes) ? true : false;
                    }
                    if(rule.any_nw_dst ^ r.any_nw_dst){
                          overlap += NW_DST_BIT;
                          whoOverlapes = (rule.any_nw_src &&
!whoOverlapes) ? true : false;
                    }

      if(rule.nw_src_prefix_and_mask.equals(r.nw_src_prefix_and_mask))
                          sameField += NW_SRC_BIT;

      if(rule.nw_dst_prefix_and_mask.equals(r.nw_dst_prefix_and_mask))
                          sameField += NW_DST_BIT;

                    if((overlap | sameField) == 127 && overlap > 0)
                          return ((whoOverlapes) ? NEW_RULE_OVERLAPS :
NEW_RULE_OVERLAPED) + r.ruleid;

                    // Check Layer 4 Overlap
                    if(rule.any_nw_proto ^ r.any_nw_proto){
                          overlap += NW_PROTO_BIT;
                          whoOverlapes = (rule.any_nw_proto &&
!whoOverlapes) ? true : false;
```

```
                  }
                  if(rule.any_tp_src ^ r.any_tp_src ){
                        overlap += TP_SRC_BIT;
                        whoOverlapes = (rule.any_tp_src &&
!whoOverlapes) ? true : false;
                  }
                  if(rule.any_tp_dst ^ r.any_tp_dst){
                        overlap += TP_DST_BIT;
                        whoOverlapes = (rule.any_tp_dst &&
!whoOverlapes) ? true : false;
                  }
                  if(rule.nw_proto.equals(r.nw_proto))
                        sameField += NW_PROTO_BIT;
                  if(rule.tp_src.equals(r.tp_src))
                        sameField += TP_SRC_BIT;
                  if(rule.tp_dst.equals(r.tp_dst))
                        sameField += TP_DST_BIT;

                  if((overlap | sameField) == 1027 && overlap > 0){
                        return ((whoOverlapes) ? NEW_RULE_OVERLAPS :
NEW_RULE_OVERLAPED) + r.ruleid;
                  }
            }
      }
      return null;
}
```