

# Implementação de um ambiente de teste SDN para gestão de QoS usando OpenFlow

Milton Baptista Semedo Lopes

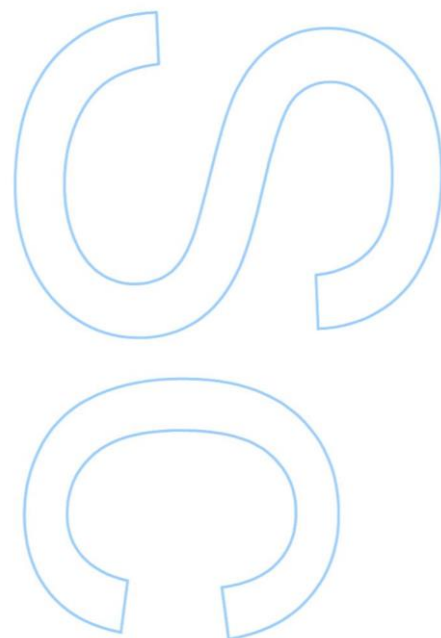
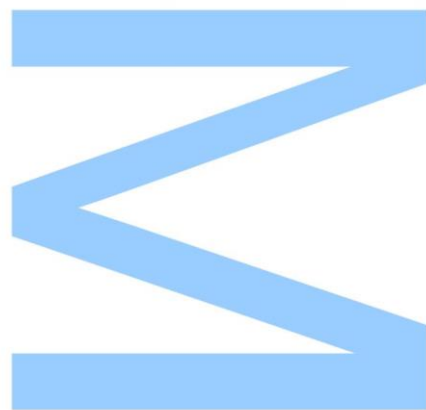
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos  
Departamento de Ciências e Computadores  
2021

## **Orientador**

Pedro Miguel Alves Brandão, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto

## **Coorientador**

Ketyllen da Costa Silva, Assistente de Investigação, Faculdade de Ciências da Universidade do Porto

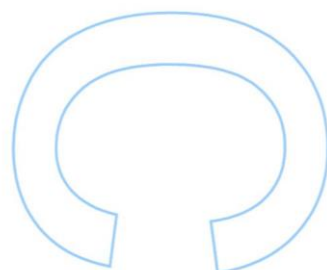
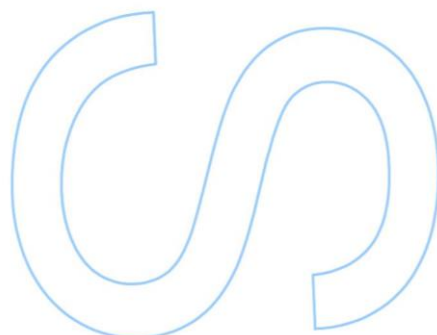
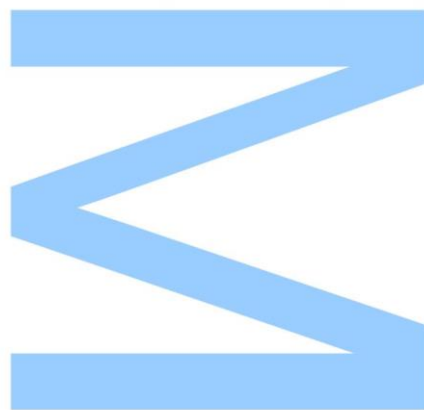




Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, \_\_\_\_/\_\_\_\_/\_\_\_\_



# Abstract

Given the limitations of the traditional Internet and the consequent increase in network flow due to the significant growth of users, it is intended to develop a Software Defined Networking (SDN) to make Quality of Service (QoS) in multimedia traffic based on the control of flows by SDN.

Thus, a general study of existing and available resources was made in order to proceed to the implementation of an SDN test environment. With the course of implementations and observations, progressed further to the implementation of a hybrid SDN network, composed of virtual switches and physical switches. Demonstrating and further leveraging the programmable and flexible features of SDN networks.

After the implementation of the SDN test environment a set of experiments and observations regarding the network operation were performed. Also taking advantage of its automated mode of operation to make the implementation of quality of service configurations in the network.

It was observed how advantageous the use of the SDN network would be in the management of network services and in responding to the limitations of the traditional Internet.

Therefore, the use of SDN network is proposed. Bearing in mind that an optimised network makes it faster and less susceptible to failures. These factors provide the emergence of new services, better use of human resources and materials and therefore better financial results.

Keywords: SDN, OpenFlow, QoS.



# Resumo

Perante as limitações da Internet tradicional e o consequente aumento de fluxo na rede devido ao crescimento significativo de utilizadores, pretende-se desenvolver uma rede definida por *software* (SDN) para fazer o QoS em tráfego de multimédia baseando-se no controle de fluxos pelo SDN.

Desta forma fez-se um estudo geral dos recursos existentes e disponíveis de modo a proceder a implementação de um ambiente de teste SDN. Com o decorrer das implementações e observações, progrediu-se ainda para a implementação de uma rede SDN híbrida, composta por comutadores virtuais e comutadores físicos. Demonstrando e aproveitando ainda mais as características programáveis e flexíveis das redes SDN.

Após a implementação do ambiente de teste SDN realizou-se um conjunto de experimentos e observações em relação ao funcionamento da rede. Aproveitando ainda do seu modo de funcionamento automatizada para fazer a implementação de configurações da qualidade de serviço na rede.

Observou-se o quão vantajoso seria a utilização da rede SDN no gerenciamento de serviços de rede e na resposta as limitações da Internet tradicional.

Assim sendo propõe-se o uso da rede SDN. Tendo em conta que, uma rede otimizada faz com que a mesma se torne mais rápida e menos suscetível a falhas. Fatores estes que proporcionam o aparecimento de novos serviços, o melhor aproveitamento dos recursos humanos e matérias e logo melhor resultado financeiro.

Palavras chaves: SDN, OpenFlow, QoS.



# Agradecimentos

Ao meu orientador Prof. Pedro Brandão e co-orientadora Ketyllen Silva pela disponibilidade demonstrada sempre durante todo o processo de desenvolvimento desta dissertação, juntamente com o incentivo constante. Obrigado, também pela orientação, aprendizagem, paciência e pela oportunidade de trabalhar neste projeto.

Obrigado, a todos os meus amigos e primos, que têm sido a minha família em todos estes anos que tenho estado longe de casa e que me ajudaram a manter o equilíbrio entre a minha vida académica e pessoal.

Um obrigado especial a minha madrinha Ângela e o meu padrinho Ângelo pelo apoio demonstrado durante este processo. E a todos que contribuíram de forma direta ou indiretamente ao término desta fase.

E por fim, um obrigado muito especial a minha mãe Andradina Moreira e o meu pai João Lopes por tudo o que providenciaram, e principalmente pelo apoio e motivação. Sem eles nada disso seria possível.

**Pa nhâ pai i nhâ mai!**

# Conteúdo

|                                                         |             |
|---------------------------------------------------------|-------------|
| <b>Abstract</b>                                         | <b>i</b>    |
| <b>Resumo</b>                                           | <b>iii</b>  |
| <b>Agradecimentos</b>                                   | <b>v</b>    |
| <b>Conteúdo</b>                                         | <b>x</b>    |
| <b>Lista de Tabelas</b>                                 | <b>xi</b>   |
| <b>Lista de Figuras</b>                                 | <b>xv</b>   |
| <b>Lista de Blocos de Código</b>                        | <b>xvii</b> |
| <b>Acrónimos</b>                                        | <b>xix</b>  |
| <br>                                                    |             |
| <b>1 Introdução</b>                                     | <b>1</b>    |
| 1.1 Objetivos . . . . .                                 | 2           |
| 1.2 Organização . . . . .                               | 2           |
| <br>                                                    |             |
| <b>2 Estado da Arte</b>                                 | <b>3</b>    |
| 2.1 Arquitetura de rede . . . . .                       | 3           |
| 2.2 Arquitetura dos comutadores . . . . .               | 4           |
| 2.3 Redes Definidas por <i>Software</i> - SDN . . . . . | 5           |
| 2.3.1 Plano de controle . . . . .                       | 6           |
| 2.3.2 Plano de Dados . . . . .                          | 6           |

|        |                                                                       |    |
|--------|-----------------------------------------------------------------------|----|
| 2.3.3  | Plano de aplicação ou gerenciamento . . . . .                         | 7  |
| 2.3.4  | Northbound Interface . . . . .                                        | 9  |
| 2.3.5  | Southbound Interface . . . . .                                        | 9  |
| 2.4    | ONOS . . . . .                                                        | 10 |
| 2.4.1  | Aplicação reativa vs Aplicação proativa . . . . .                     | 10 |
| 2.4.2  | Encaminhamento de pacotes . . . . .                                   | 11 |
| 2.4.3  | Reactive Forwarding . . . . .                                         | 11 |
| 2.4.4  | Intent Framework . . . . .                                            | 13 |
| 2.4.5  | Intent-based reactive forwarding . . . . .                            | 15 |
| 2.5    | OpenFlow . . . . .                                                    | 17 |
| 2.5.1  | Mensagens OpenFlow . . . . .                                          | 20 |
| 2.5.2  | Falha na ligação comutador e controlador . . . . .                    | 23 |
| 2.6    | Mininet . . . . .                                                     | 24 |
| 2.7    | Open vSwitches . . . . .                                              | 25 |
| 2.8    | Topologias de Redes SDN . . . . .                                     | 26 |
| 2.8.1  | Topologia simples . . . . .                                           | 26 |
| 2.8.2  | Um controlador para vários comutadores . . . . .                      | 26 |
| 2.8.3  | Vários controladores para vários comutadores . . . . .                | 27 |
| 2.8.4  | Controlador Distribuído . . . . .                                     | 27 |
| 2.8.5  | Arquitetura para Internet . . . . .                                   | 29 |
| 2.8.6  | Controladores centralizados e distribuídos . . . . .                  | 29 |
| 2.9    | Tráfego multimídia . . . . .                                          | 30 |
| 2.10   | Voz sobre IP . . . . .                                                | 30 |
| 2.11   | Principais diferenças entre a rede tradicional e a rede SDN . . . . . | 31 |
| 2.12   | Qualidade de Serviço . . . . .                                        | 31 |
| 2.12.1 | DiffServ . . . . .                                                    | 33 |
| 2.12.2 | AES67 . . . . .                                                       | 34 |
| 2.12.3 | Queues . . . . .                                                      | 37 |

|          |                                                                        |           |
|----------|------------------------------------------------------------------------|-----------|
| 2.12.4   | As vantagens da rede SDN para tráfego de voz sobre IP . . . . .        | 37        |
| <b>3</b> | <b>Ambiente de teste SDN</b>                                           | <b>39</b> |
| 3.1      | Configuração do controlador . . . . .                                  | 39        |
| 3.2      | Configuração comutadores HP Aruba . . . . .                            | 40        |
| 3.3      | Configuração OpenFlow . . . . .                                        | 41        |
| 3.4      | <i>intent</i> proativo . . . . .                                       | 43        |
| 3.4.1    | Corte de ligação direta . . . . .                                      | 45        |
| 3.4.2    | Ligação dupla entre comutadores . . . . .                              | 45        |
| 3.5      | Ligação rede virtual e física . . . . .                                | 45        |
| 3.6      | Configuração da ligação entre a rede física e a rede virtual . . . . . | 47        |
| 3.6.1    | Experimento 1: Implementação . . . . .                                 | 48        |
| 3.6.2    | Experimento 2: Estudo das apps . . . . .                               | 50        |
| 3.6.3    | Experimento 3: Configuração de VLANs . . . . .                         | 53        |
| 3.6.4    | Experimento 4: Configuração OpenFlow Packet-in message . . . . .       | 55        |
| 3.7      | Topologia da rede híbrida . . . . .                                    | 55        |
| <b>4</b> | <b>Implementação de QoS na rede</b>                                    | <b>59</b> |
| 4.1      | Configurações nos comutadores HP Aruba . . . . .                       | 59        |
| 4.2      | Configurações nas Open vSwitches . . . . .                             | 64        |
| 4.3      | Qualidade de serviço na rede implementada . . . . .                    | 64        |
| 4.4      | Injeção de tráfego . . . . .                                           | 65        |
| 4.5      | QoS: Intent + setDSCP . . . . .                                        | 66        |
| 4.6      | QoS na rede virtual: Intent + setQueue . . . . .                       | 68        |
| 4.7      | QoS rede híbrida . . . . .                                             | 70        |
| <b>5</b> | <b>Conclusões</b>                                                      | <b>75</b> |
| 5.1      | Trabalho Futuro . . . . .                                              | 77        |
|          | <b>Bibliografia</b>                                                    | <b>79</b> |

|          |                                           |           |
|----------|-------------------------------------------|-----------|
| <b>A</b> | <b>Configurações comutadores HP Aruba</b> | <b>83</b> |
| <b>B</b> | <b>Scripts para emulação da rede</b>      | <b>87</b> |

# Lista de Tabelas

|     |                                                               |    |
|-----|---------------------------------------------------------------|----|
| 2.1 | Mensagens OpenFlow [30]                                       | 22 |
| 2.2 | Rede SDN vs Rede Tradicional [28]                             | 32 |
| 2.3 | Classes de QoS e DiffServ                                     | 35 |
| 3.1 | Tabela de estudo sobre a influências de apps na ligação       | 50 |
| 3.2 | Tabela de estudo sobre a influências da regra IPV4 na ligação | 52 |
| 3.3 | Resumo dos testes com VLAN 1                                  | 56 |
| 3.4 | Resumo dos testes com VLAN 2                                  | 57 |



# Lista de Figuras

|      |                                                                  |    |
|------|------------------------------------------------------------------|----|
| 2.1  | Arquitetura dos comutadores tradicionais[16]                     | 4  |
| 2.2  | Rede Tradicional vs. Rede SDN                                    | 5  |
| 2.3  | Arquitetura de rede SDN                                          | 8  |
| 2.4  | Subsistemas ONOS                                                 | 10 |
| 2.5  | Reactive Forwarding                                              | 12 |
| 2.6  | onos-fwd-app                                                     | 13 |
| 2.7  | Compilação do <i>intent</i>                                      | 14 |
| 2.8  | Intent Forwarding                                                | 16 |
| 2.9  | onos-ifwd-app                                                    | 16 |
| 2.10 | Funcionamento do protocolo OpenFlow                              | 17 |
| 2.11 | Arquitetura OpenFlow                                             | 18 |
| 2.12 | Comutador OpenFlow                                               | 19 |
| 2.13 | Cabeçalho OpenFlow[5]                                            | 19 |
| 2.14 | Tabela de Fluxos                                                 | 20 |
| 2.15 | Comunicação In-band e Out-of-band[5]                             | 21 |
| 2.16 | Visão geral do Mininet[11]                                       | 24 |
| 2.17 | Open vSwitch[25]                                                 | 25 |
| 2.18 | Topologia simples rede SDN                                       | 26 |
| 2.19 | Um controlador para vários comutadores                           | 27 |
| 2.20 | Um controlador para vários comutadores conectados entre si       | 27 |
| 2.21 | Vários controladores para vários comutadores conectados entre si | 28 |

|      |                                                                                        |    |
|------|----------------------------------------------------------------------------------------|----|
| 2.22 | Controlador Distribuído . . . . .                                                      | 28 |
| 2.23 | Arquitetura para Internet . . . . .                                                    | 29 |
| 2.24 | Interoperabilidade do padrão AES67 . . . . .                                           | 31 |
| 3.1  | ONOS GUI . . . . .                                                                     | 42 |
| 3.2  | Topologia da rede implementada . . . . .                                               | 43 |
| 3.3  | <i>intent</i> máquina-a-máquina . . . . .                                              | 44 |
| 3.4  | Falha na ligação entre comutadores . . . . .                                           | 45 |
| 3.5  | Falha na ligação dupla entre comutadores . . . . .                                     | 46 |
| 3.6  | Rede física sem ligação com rede virtual . . . . .                                     | 47 |
| 3.7  | Topologia de rede incluindo a ligação entre a rede virtual e a rede física . . . . .   | 49 |
| 3.8  | Print de resposta arp enviado pelo host físico, capturado no host virtual H1 . . . . . | 52 |
| 3.9  | Configuração OpenFlow packet-in HP Aruba Switch . . . . .                              | 58 |
| 3.10 | Topologia de rede híbrida . . . . .                                                    | 58 |
| 4.1  | Configuração PTP - HP Aruba Switch . . . . .                                           | 60 |
| 4.2  | Configuração Queues - HP Aruba Switch . . . . .                                        | 60 |
| 4.3  | Largura de banda de cada queue - HP Aruba Switch . . . . .                             | 61 |
| 4.4  | QoS Type of Service - HP Aruba Switch . . . . .                                        | 61 |
| 4.5  | QoS dscp-map - HP Aruba Switch . . . . .                                               | 62 |
| 4.6  | Port Trust Mode - HP Aruba Switch . . . . .                                            | 63 |
| 4.7  | Caminho <i>intent</i> + DSCP . . . . .                                                 | 66 |
| 4.8  | Tabela de fluxos comutador HP . . . . .                                                | 67 |
| 4.9  | HP-1 - acção setDSCP . . . . .                                                         | 68 |
| 4.10 | Configuração <i>intent</i> + setQueue . . . . .                                        | 69 |
| 4.11 | Estatísticas Queues porta s3-eth4 . . . . .                                            | 69 |
| 4.12 | Rede Híbrida + <i>intent</i> . . . . .                                                 | 70 |
| 4.13 | Largura de Banda e Jitter de tráfego sem prioridade 1.0 . . . . .                      | 71 |
| 4.14 | Largura de Banda e Jitter de tráfego sem prioridade 2.0 . . . . .                      | 72 |

|      |                                                                   |    |
|------|-------------------------------------------------------------------|----|
| 4.15 | Largura de Banda e Jitter de tráfego com prioridade 3.0 . . . . . | 73 |
| 4.16 | Propagação de tráfego . . . . .                                   | 73 |
| A.1  | Instância aggregate inicial . . . . .                             | 84 |
| A.2  | Regra de fluxo do comutador HP para Openflow 1.0 . . . . .        | 84 |
| A.3  | Regra de fluxo do comutador HP para Openflow 1.3 . . . . .        | 85 |



# Lista de Blocos de Código

|     |                                                                                |    |
|-----|--------------------------------------------------------------------------------|----|
| 3.1 | Adicionar interface física dentro da <i>bridge</i> virtual . . . . .           | 48 |
| 3.2 | Configurar as <i>bridges</i> das switches para funcionamento in-band . . . . . | 48 |
| 3.3 | Configurar <i>connection-mode</i> de cada OVS para in-band . . . . .           | 49 |
| B.1 | Script de emulação da rede em Python – Mininet . . . . .                       | 88 |
| B.2 | Script de emulação da rede em Python – Mininet – Host enviando tags . . . . .  | 89 |
| B.3 | Script de configuração da ligação – com interface física em Vlan . . . . .     | 89 |
| B.4 | Script de configuração da ligação da rede virtual com rede física . . . . .    | 90 |
| B.5 | Script queueConfig . . . . .                                                   | 91 |



# Acrónimos

|                 |                                          |              |                                           |
|-----------------|------------------------------------------|--------------|-------------------------------------------|
| <b>AES</b>      | Audio Engineering Society                | <b>MPLS</b>  | Multi Protocol Label Switching            |
| <b>API</b>      | Application Programming Interface        | <b>NOS</b>   | Network Operating System                  |
| <b>ASIC</b>     | Application Specific Integrated Circuits | <b>OF</b>    | OpenFlow                                  |
| <b>App</b>      | Aplication                               | <b>ONF</b>   | Open Networking Foundation                |
| <b>CoS</b>      | Class of Service                         | <b>OVS</b>   | Open vSwitches                            |
| <b>CLI</b>      | Command Line Interface                   | <b>OVSDB</b> | Open vSwitch Database Management Protocol |
| <b>DSCP</b>     | Differentiated Services Code Point       | <b>PHB</b>   | Per-hop behavior                          |
| <b>DS</b>       | Differentiated Services                  | <b>PTP</b>   | Precision Time Protocol                   |
| <b>DiffServ</b> | Differentiated Service                   | <b>QoS</b>   | Quality of Service                        |
| <b>GUI</b>      | Graphical User Interface                 | <b>ToS</b>   | Type of Service                           |
| <b>HP</b>       | Hewlett packard                          | <b>TCP</b>   | Transmission Control Protocol             |
| <b>ICMP</b>     | Internet Control Message Protocol        | <b>UDP</b>   | User Datagram Protocol                    |
| <b>IETF</b>     | Internet Engineering Task Force          | <b>VLAN</b>  | Virtual Local Area Network                |
| <b>IntServ</b>  | Integrated Service                       | <b>SDN</b>   | Software Defined Networking               |
| <b>IPv4</b>     | Internet Protocol Version 4              |              |                                           |



# Capítulo 1

## Introdução

Ao longo do tempo a Internet tem evoluído de forma expressiva impulsionada pela crescente demanda na utilização de serviços e aplicações. Como consequência, têm-se o aumento no número de novos utilizadores e o acentuado fluxo de tráfego na rede [27]. A exemplo, temos atual situação pandêmica gerada pela COVID-19, o tráfego na rede teve um aumento de cerca de 40% a nível mundial [10]. Contudo os utilizadores desejam cada vez mais inovações nos serviços e na performance rápida destes serviços.

A indústria de áudio e tráfego multimídia também vem se deparando com um aumento significativo de dados para processar e transportar, forçando-lhes a procura de novas soluções [3].

Com cada vez mais exigências de qualidade de serviço, e com o aumento e facilidade de ligar dispositivos à rede, as limitações da rede tradicional local têm-se intensificado. Os usuários querem cada vez mais filmes de alta definição, interfaces mais avançadas para seus aplicativos e sites conduzindo ao aumento significativo na demanda por largura de banda. Evidenciando a necessidade de projetar uma infraestrutura de rede capaz de suportar essas demandas.

Há ainda alguns desafios na implementação da rede, como por exemplo ao adicionar novos *hardwares*, como comutadores e roteadores estes ainda necessitam ser configurados manualmente via *Command Line Interface (CLI)*, o que suscita a erros durante o procedimento.

Ao considerar a escalada quanto ao número de dispositivos, têm-se problemas ainda maiores como a complexidade da rede, o tempo gasto nas configurações, a limitação de potencial da rede, sem contar com os custos financeiros associados.

A falta de escalabilidade e flexibilidade da rede tradicional tem sido um grande problema, uma vez que este está a limitar os serviços e não evoluir como esperado [27]. Face a estes problemas, o paradigma SDN surge como solução.

Com a rede SDN torna-se mais fácil implementar e testar novos protocolos de redes, possibilitando uma evolução mais rápida das tecnologias de redes devido as suas características programáveis e flexíveis, diminuindo assim, a dificuldade de implementação de novas tecnologias.

Deste modo pretende-se desenvolver e implementar uma rede SDN de maneira a fazer um estudo sobre a performance do tráfego de áudio sobre a rede.

## 1.1 Objetivos

O objetivo principal deste trabalho consiste em projetar, desenvolver e testar uma *Software Defined Networking (SDN)* para avaliar o *Quality of Service (QoS)* em tráfego de dados multimídia.

Tem-se como objetivos específicos os seguintes pontos:

- Projetar uma rede de teste SDN para controle de tráfego multimídia com principal foco no tráfego de voz sobre IP;
- Implementar a rede funcional utilizando comutadores físicos disponibilizados da HP Aruba, juntamente com a escolha e configuração de um controlador OpenFlow e estações de trabalho;
- Aumentar a escalabilidade da rede, configurando uma rede híbrida composta por dispositivos físicos e dispositivos virtuais;
- Investigar a qualidade de serviço na rede implementada com base nas recomendações de QoS para o tráfego de voz sobre IP;

## 1.2 Organização

O resto da dissertação encontra-se organizada da seguinte forma:

- No capítulo 2 é descrito o estado da arte;
- No capítulo 3 é descrita a implementação do ambiente SDN;
- No capítulo 4 são descritas as configurações efetuadas para se obter a qualidade de serviço na rede implementada e a sua avaliação;
- No capítulo 5 são descritas as discussões sobre as realizações implementadas, limitações e orientações para trabalho futuro;

## Capítulo 2

# Estado da Arte

### 2.1 Arquitetura de rede

No modelo tradicional de arquitetura de redes de computadores, todos os dispositivos que constituem a rede, desde os roteadores, comutadores, e pontos de acesso são compostos pelos planos de controle e plano de dados, estes encontram-se unificados dentro de cada um destes dispositivos [31].

O plano de controle é o responsável pela configuração do dispositivo em particular em que este está inserido, ou seja, ele faz a configuração dos caminhos que serão utilizados pelo plano de dados do mesmo dispositivo. Tendo esses caminhos determinados, estes são inseridos na tabela de transmissão do plano de dados, que é o responsável pela transmissão e reencaminhamento de fluxos [22].

No modelo tradicional, uma vez que a tabela de transmissão tenha sido configurada pela camada de controle, a única maneira de um administrador de rede fazer mudanças na política de reencaminhamento é através de configurações físicas realizadas no dispositivo.

A flexibilidade é um dos obstáculos recorrente para o modelo de rede tradicional. Existem poucas APIs disponíveis para provisionamento e a maioria dos dispositivos de comutação são proprietários. Apesar das redes tradicionais funcionarem bem com os *softwares* de provisionamento proprietários, estes não permitem uma alteração rápida conforme as necessidades da rede.

As principais características dos modelos de rede tradicional consistem em, as suas funções são normalmente executados pelos dispositivos dedicados como os comutadores e roteadores, a sua funcionalidade baseia-se na implementação em hardware dedicado, como os Application Specific Integrated Circuits (ASIC). Contudo devido a estas características específicas estes vêm apresentando muitas limitações [17].

## 2.2 Arquitetura dos comutadores

Os comutadores possuem tradicionalmente uma estrutura composta por três camadas. A camada de controle, a camada de gerenciamento e a camada de dados como apresentada na figura 2.1. Estes comunicam-se de forma horizontal ou vertical com as outras camadas.

A camada de dados é a responsável pela memória, pela transmissão, pelo reencaminhamento e pela modificação de cabeçalhos dos pacotes se necessário. Ela é composta pela tabela de encaminhamento e por diferentes portas que são utilizadas na recepção e na transmissão de pacotes. Quando a informação no cabeçalho de um pacote é encontrada na tabela de encaminhamento, esta pode ser modificada ou não e depois encaminhada sem nenhuma interferência das outras camadas [30].

O plano de controle tem como principal função manter atualizada a tabela de transmissão na camada de dados, de modo que a camada de dados consiga processar de forma independente o maior quantidade de tráfego possível. Este também é o responsável pelo processamento de vários protocolos de controle que podem ter impacto na tabela na transmissão dependendo das configurações e do tipo de comutador.

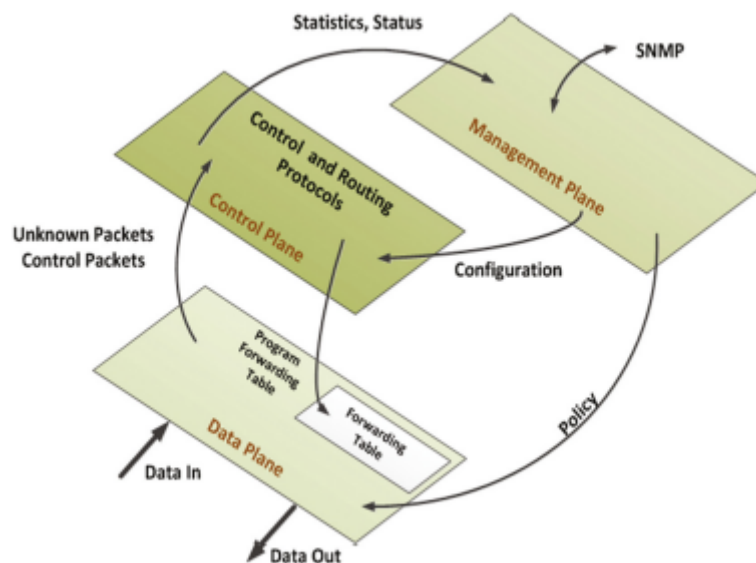


Figura 2.1: Arquitetura dos comutadores tradicionais[16]

No que diz respeito à camada de gerenciamento, esta é a camada aonde os administradores de redes realizam as configurações necessárias e fazem o monitoramento da performance do comutador, que por sua vez, consiste em extrair informação ou modificar dados nos planos de dados e de controle [16].

## 2.3 Redes Definidas por *Software* - SDN

As redes definidas por *software* são uma abordagem emergente que consiste na separação física entre o plano de controle e o plano de dados. A sua utilização tem estado em constante crescimento devido aos seus aspetos dinâmicos, e suas características quanto a programabilidade, gestão centralizada, configurado programaticamente e com base em padrões abertos [14]. Desta forma, as redes SDNs permitem o controle da rede de maneira central e global o que é interessante para a natureza dinâmica das aplicações de hoje.

A separação entre os dois planos, possibilita a configuração do plano de controle, sem ser o do fabricante, como representada na figura 2.2. No entanto, para tal é necessário estabelecer a forma como os planos de controle e de dados comunicam entre si e a interface utilizada pelo plano de controle para programar o comportamento do plano de dados. Interface esta, que geralmente é a *Application Programming Interface (API)* do controlador.

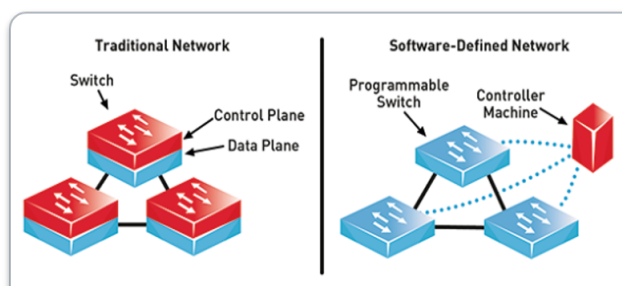


Figura 2.2: Rede Tradicional vs. Rede SDN

A arquitetura proposta para as redes SDN, é composta por três camadas principais conforme pode-se observar na figura 2.3 e as interfaces de comunicação entre elas. O plano de controle é implementado pelo controlador, que oferece uma visão centralizada da rede global e permite que os administradores determinem aos equipamentos subjacentes como o plano de dados deve lidar com o tráfego de rede.

O controlador interage com a camada de aplicação, o qual permite ao administrador da rede configurar o tráfego e implantar serviços de forma programática. A interface de comunicação entre o controlador e essas aplicações é designada de “*Northbound interface*”.

O plano de dados é implementado por equipamentos como comutadores e roteadores que permitem a conexão com o controlador. A interface de comunicação entre o controlador e os comutadores é designada de “*Southbound interface*” [30].

A rede SDN visa ultrapassar as limitações da rede tradicional, caracterizada pela alta complexibilidade no que tange ao gerenciamento descentralizada, satisfazendo as novas aplicações que exigem mais flexibilidade e fácil solução de problemas.

O modelo de rede definida por *software* veio adicionar uma nova abordagem a gestão de rede utilizada atualmente. Este é definido pela separação dos planos de controle e planos de dados.

Permite que as redes se liguem diretamente as aplicações através das APIs, providenciando deste modo mais meios para o melhor desempenho e segurança das aplicações, juntamente com uma arquitetura de rede flexível e dinâmica que pode ser alterada de acordo com as necessidades da rede [7].

A rede SDN possibilita a gestão centralizada, o controle programático, a otimização dos recursos de rede, juntamente com uma API aberta que permite manter o controle da rede.

### 2.3.1 Plano de controle

O plano de controle consiste na inteligência da rede. Composto pelo *Network Operating System (NOS)* este permite um controle centralizado logicamente da rede. Podendo estabelecer mais do que um único controlador para a rede. Ele fornece abstrações e APIs aos administradores de rede, possibilitando-os uma definição de política de rede simplificada e com menores possibilidades de erro.

O controlador tem como principais funções:

- Encaminhar dados pelo caminho mais curto e mais adequada ao tráfego em análise;
- Fornecer informações sobre a topologia;
- Descoberta de novos dispositivos conectados na rede;
- Análise do estado da rede;
- Distribuição de configurações da rede
- Mecanismos de segurança e gerenciamento de dispositivos;

Contudo, de modo a executar as suas funções, o sistema operacional do controlador possui instalada um conjunto de protocolos, aplicações e *softwares* que permitem a realização das suas tarefas por toda a rede.

### 2.3.2 Plano de Dados

A principal função do plano de dados é realizar a transmissão e encaminhamento dos tráfegos da rede. Composto pelos dispositivos de encaminhamento, como *hardwares* de comutadores, roteadores e também por *softwares*, como máquinas e comutadores virtuais cuja função é encaminhar os tráfegos da rede. Contudo estes dispositivos diferem dos comutadores e roteadores convencionais, pois estes não possuem implementações de protocolos e programas complexos. Toda inteligência encontra-se no plano de controle.

Por exemplo, num dispositivo com suporte ao protocolo OpenFlow, apenas são instaladas as tabelas de encaminhamento, que contêm as ações a serem tomadas para cada fluxo de pacotes

que chegam no dispositivo. Um fluxo é definido com um conjunto de pacotes com características semelhantes.

As tabelas de encaminhamento, são compostas por regras de fluxos que comparam os valores nos campos de cabeçalho de cada pacote, como por exemplo, o endereço de destino e partida, o protocolo, o número de porta e a prioridade com valores já definidos na tabela. Tendo como resultado a execução da ação associada a regra, ou a ação associada no caso a regra falha.

Dentre as ações que podem ser realizadas pelo plano de dados para cada tráfego estão:

- Encaminhar o pacote para uma porta de saída;
- Encapsular e encaminhar o pacote para o controlador;
- Descartar o pacote;
- Enviar para a próxima tabela de fluxos ou para alguma tabela especial;
- Enviar para o encaminhamento normal do equipamento nas camadas 2 (Internet) ou 3 (Transporte)

Caso as regras falhem, o pacote pode ser descartado, ou enviado para o controlador, isto fica a critério do administrador de rede que pode definir uma regra padrão para tratar dos tráfegos que não satisfazem as outras regras.

### 2.3.3 Plano de aplicação ou gerenciamento

O plano de aplicação ou gerenciamento, constitui uma parte importante da rede SDN. Pois é a interface entre uma rede SDN e o administrador de rede. Torna-se possível programar os equipamentos de redes de acordo com as funções pretendidas, além de obter informações sobre o estado da rede e eventos que ocorrem. Essas configurações são realizadas de forma fácil devido, às APIs que executam a comunicação entre o plano de controle e o plano de aplicação, as linguagens de programação usadas nas aplicações que fornecem várias abstrações e facilitam a configuração e reuso do código, além de evitar conflitos entre comandos vindos de várias aplicações.

Um dos melhores benefícios do uso da rede SDN é a sua capacidade em suportar e habilitar a implementação de aplicações (apps) inovadores. As apps SDN são implementadas no plano superior ao plano do controlador e são contempladas com uma visão abstrata dos recursos da rede providenciadas pelo controlador através do Northbound API [30].

As apps SDN são numa análise mais aprofundada os responsáveis pela gestão dos fluxos de redes dos dispositivos. Utilizando a API do controlador para fazer esse gerenciamento elas podem configurar fluxos para encaminhar pacotes através do menor caminho, balancear o tráfego através de múltiplos caminhos ou destinadas a um conjunto de *end-points*, reagir a alterações na topologia de rede como a falha nas ligações ou introdução de novos dispositivos e caminhos,

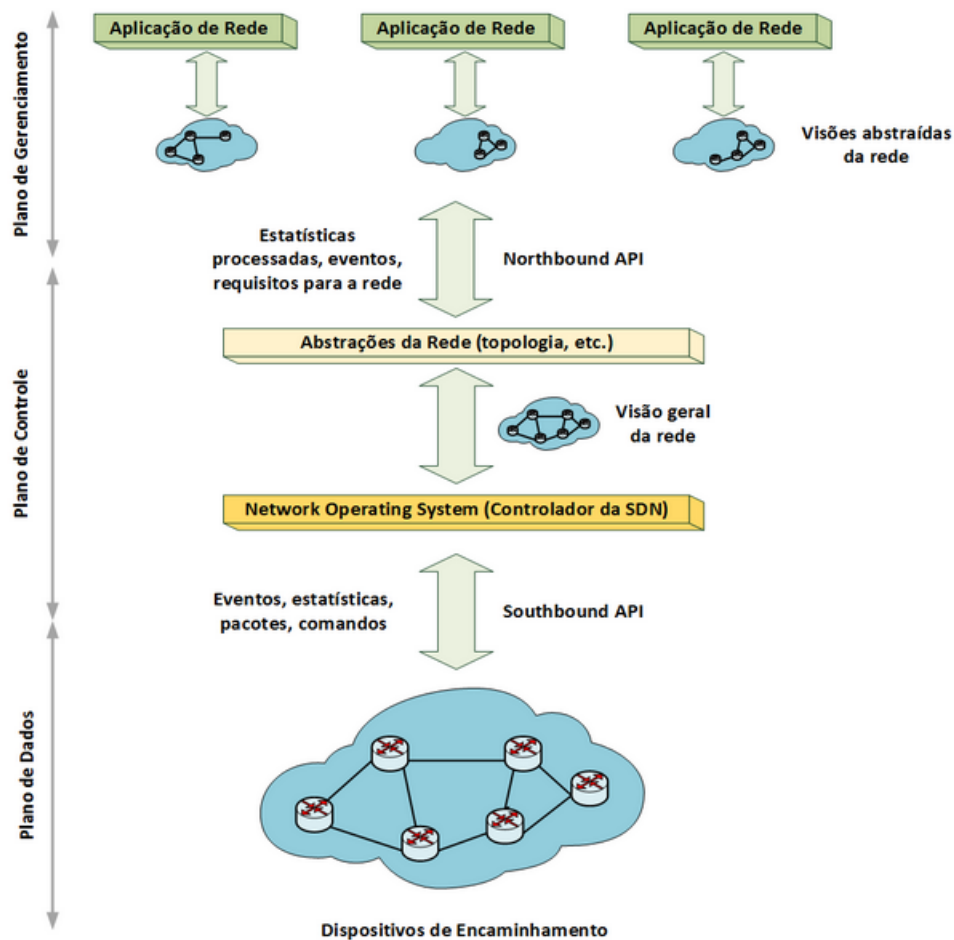


Figura 2.3: Arquitetura de rede SDN

redirecionar o tráfego para propósitos de inspeções, autenticação ou outras tarefas de segurança. Consequentemente, o valor da arquitetura SDN torna-se altamente expansível, uma vez que as decisões de encaminhamento e roteamento podem ser controladas por uma gama mais rica de apps que controlam o controlador.

### 2.3.3.1 Northbound API para aplicações

A Northbound API é aquela que coloca as aplicações no controle da rede, ou seja, em vez de ajustar a infraestrutura de rede repetidamente para fazer um serviço funcionar corretamente, uma estrutura pode ser configurada para permitir que a aplicação possa exigir a infraestrutura que precisa. APIs Northbound geralmente fornecem uma lista de funções de rede base relacionadas ao fornecedor, que são então usadas para configurar a infraestrutura específica do usuário, e o controlador a interpreta em uma linguagem que cada nó de infraestrutura pode entender. Permitindo assim a implementação de novas aplicações.

A principal razão do aparecimento do Northbound API é a necessidade de aplicações ou

sistemas de gerenciamento externos em extrair informações do plano de controle. É usado para a inovação e ter um melhor arranjo eficaz da rede. Tornando-se necessários para o alinhamento da rede com as necessidades de diferentes aplicações, permitindo a máxima utilidade da arquitetura SDN.

Por vezes considerado o APIs mais crítico num ambiente SDN. Deve-se ao facto do valor do SDN estar associado a aplicativos inovadores que ele pode suportar e implementar. Com isso o controlador não se torna numa ferramenta estática, podendo interagir com sistemas separados e fornecer abstrações da rede. É de salientar que o Northbound API deve ser aberto e padronizado a fim de promover a portabilidade e interoperabilidade das aplicações.

#### 2.3.4 Northbound Interface

A *Northbound Interface* é a ponte que permite a comunicação entre o plano de aplicação ou gerenciamento e o plano de controle, sendo assim uma fração importante da arquitetura SDN. Desta forma ela permite que as aplicações utilizadas pelos administradores de redes executem o controle e monitoramento da rede sem a necessidade de outras configurações.

As *APIs Northbound* providenciadas variam do tipo de controlador utilizado. Contudo têm como principais funções traduzir os requisitos das aplicações de gerenciamento, em instruções de baixo nível para os dispositivos da rede e transferir estatísticas que foram geradas nos dispositivos e processados pelo controlador.

#### 2.3.5 Southbound Interface

A *Southbound Interface* é a ponte que permite a comunicação entre o plano de controle e o plano de dados. Ele é um elemento vital na separação entre os dois planos. Sendo também uma API, é através dela que os controladores enviam as instruções de configurações das aplicações para a rede, reprogramando os dispositivos de acordo com as necessidades, como por exemplo, firewall, roteamento e controle de fluxo. Estas instruções de configurações são feitas adicionando ou removendo regras de fluxos nas tabelas de encaminhamento.

*Southbound Interface* também é utilizada pelos dispositivos de redes, sempre que houver necessidade de comunicarem com o controlador. Isto ocorre nos seguintes casos:

- A necessidade de comunicar o acontecimento de algum evento na rede, como mudanças de porta.
- No envio de estatísticas de tráfego e outras informações detalhadas da rede para o controlador, fornecendo assim características da rede para os administradores de rede.
- No envio de pacotes para o controlador. Isto acontece em dois casos. O primeiro, quando os dispositivos do plano de dados não têm instruções para o tratamento do pacote, isto é,

quando não se encontram instaladas regras de fluxos nos dispositivos que mapeiam com o pacote. O segundo acontece quando alguma regra de fluxo instalada nos dispositivos tem como ação o envio do pacote para o controlador.

## 2.4 ONOS

O ONOS que na sua extensão significa Open Network Operating System, é um controlador da rede SDN de código aberto, que possibilita novas capacidades de rede SDN a dispositivos com serviços proprietários. Foi concebido para proporcionar um controlo a larga escala, com alto desempenho e com suporte a redes híbridas. Ele age sobre a rede SDN, controlando todos os seus componentes. O ONOS e os seus pacotes constituintes, são descritos em Java, e permitem a sua configuração personalizada e desenvolvimento de novas funcionalidades. Fornece também o seu acesso e configuração a partir da GUI ou CLI.

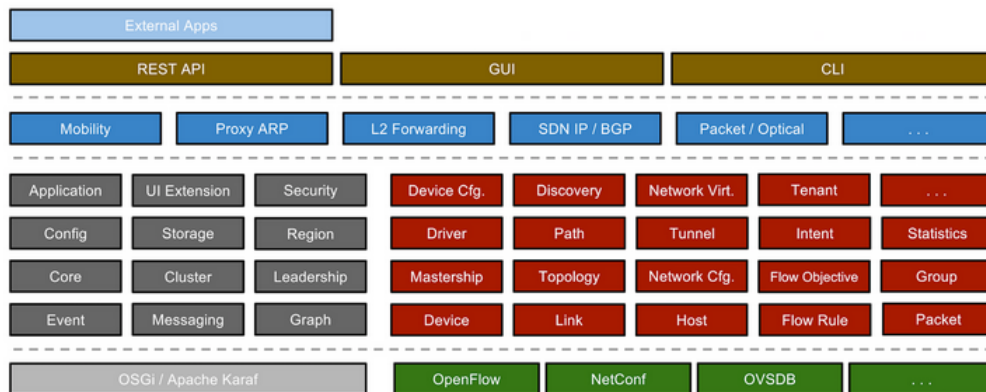


Figura 2.4: Subsistemas ONOS

O ONOS é composto por vários subsistemas, aonde cada subsistema é um conjunto de elementos que juntos compõem um serviço. Alguns dos serviços providenciados são, o subsistema de dispositivos, o subsistema de ligações, o subsistema de topologia, o subsistema de *intents*, o subsistema de regras de fluxos, o subsistema de pacotes e os outros representados na figura 2.4.

### 2.4.1 Aplicação reativa vs Aplicação proativa

A principal diferença entra essas duas abordagens é que uma app reativa vai receber periodicamente pacotes vindos dos comutadores para ser processado. Portanto envia instruções de volta ao comutador em como tratar estes pacotes e como fazer mudanças na tabela de fluxos se necessário. Nesses casos a comunicação entre o comutador e o controlador é proporcional ao número de novos pacotes recebidos na rede. Deve-se ter em conta o tempo de inatividade das regras fluxos, porque tendo um idle time pequeno resulta em poucas regras de fluxos na tabela, o que leva ao aumento no número de fluxos de tráfego sem *match* na tabela. Os aplicativos que usam essa abordagem são aqueles que precisam ver e responder aos novos fluxos criados.

Enquanto numa app proativa existe muito menos comunicação entre o comutador e o controlador porque ela já pré-configura os comutadores da rede com tabelas de fluxos ou atributos de configuração apropriadas para a gestão do tráfego de entrada mesmo antes deste chegar ao comutador.

A abordagem reativa pode ser mais suscetível a falhas em caso de perda de conectividade com o controlador. O comutador tem mais probabilidades de continuar o seu funcionamento na abordagem proativa, visto que já tem as entradas de fluxos pré-configuradas. Outro ponto positivo na abordagem proativa é que neste não há latência adicional na rede para a configuração das regras de fluxos. Contudo tem como desvantagem a diminuição da granularidade de controle da rede uma vez que os fluxos estarão agrupados.

### 2.4.2 Encaminhamento de pacotes

No ONOS, sendo o controlador responsável pelas decisões no plano de controle da rede, podemos observar diferenças na frequência de fluxos, latência, encaminhamento ou descartar de pacotes nos seus dois modos de encaminhamento.

No modo reativo, sempre que um fluxo de tráfego novo chega nos dispositivos estes o reencaminham para o controlador para mais instruções. Neste caso a aplicação instrui o controlador em como responder ao pacote recebido e se fizer sentido estabelecerá novas regras de fluxos no dispositivo de modo a permiti-lo responder localmente na próxima vez que receber um pacote do mesmo fluxo. Desta maneira, possibilita a flexibilidade na gestão de políticas de roteamento de tráfego desconhecido, e determinar outras funções de encaminhamento.

No modo proativo, os fluxos são estabelecidos pelas aplicações utilizando o controlador. Normalmente as apps estabelecem esses fluxos no momento ao serem iniciadas e estes permanecem até alguma mudança de configuração for feita. Estes são conhecidos como regras de fluxos estáticos. Outro tipo de fluxo estático é quando o controlador decide mudar o fluxo baseando na carga de tráfego de algum dispositivo da rede. Logo, neste modo os fluxos de entrada dependem das regras de tráfego pré-configuradas nos comutadores pelo controlador, exigindo um conhecimento prévio das regras de encaminhamento que podem ser aplicáveis na rede [30].

### 2.4.3 Reactive Forwarding

Utilizando a programação de fluxo salto a salto o *reactive forwarding* faz a instalação de cada novo fluxo recebido pelo controlador em toda a rede. O ONOS nos fornece a *onos-fwd-app* como a aplicação padrão para o encaminhamento reativo.

Na figura 2.5 um pacote novo segue em direção ao host 2, sendo primeiro encaminhado ao comutador 1 pelo host 1. Uma vez que o novo pacote não possui nenhuma regra de fluxo configurada, o comutador 1 envia uma mensagem `PKT_IN` ao controlador para mais instruções. O controlador processa o novo pacote e envia-a de volta ao comutador 1 como uma mensagem

PKT\_OUT que inclui ações como descartar ou encaminhar o pacote para uma porta específica.

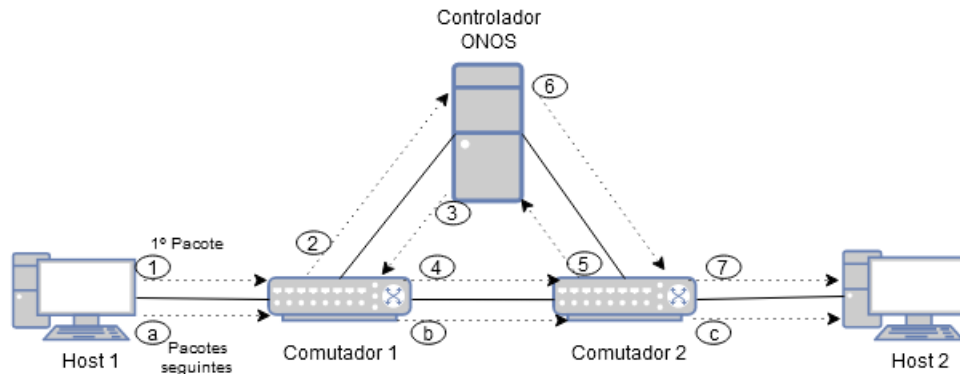


Figura 2.5: Reactive Forwarding

O controlador também envia uma mensagem FLOW-MOD para configurar uma nova regra de fluxo no comutador. Similarmente também há trocas de mensagens PKT\_IN, PKT\_OUT e FLOW\_MOD quando o comutador 2 receber o pacote. Esta troca configura a regra necessária para o pacote chegar o destino Host 2.

#### 2.4.3.1 onos-fwd-app

A app onos-fwd-app como apresentada na figura 2.6 é constituída por 4 classes Javas, a classe pública *Reactive Forwarding* e dentro dela as classes privadas *ReactivePacketProcessor*, *InternalTopologyListener* e *SrcDstPair*.

O *InternalTopologyListener* é o responsável por observar a topologia da rede e recolher todos os eventos responsáveis pela alteração na sua topologia, de seguida após a recolha, ele itera pela sua lista de eventos de modo a observar se alguns deles estão criando Buraco Negro. Buracos Negros aparecem quando são enviados pacotes para um destinatário que não se encontra online ou conectado, levando ao *dropping* desses pacotes sem informar o remetente. Deste modo essa classe tenta identificar e corrigir essas situações.

O *ReactivePacketProcessor* é o responsável por encaminhar pacotes ao longo dos seus caminhos. De início ele verifica se o pacote já foi tratado anteriormente, caso este já for processado é feito um *return*. Caso contrário ele faz uma série de testes para verificar se existe alguma configuração que não permita o tratamento do determinado pacote. Se não existir nenhum impedimento ele encontra um caminho para o destino e instala a nova regra de fluxo no comutador. O *SrcDstPair* é uma classe *wrapper* para um par de endereços MAC de origem e destino. Não faz nenhuma implementação.

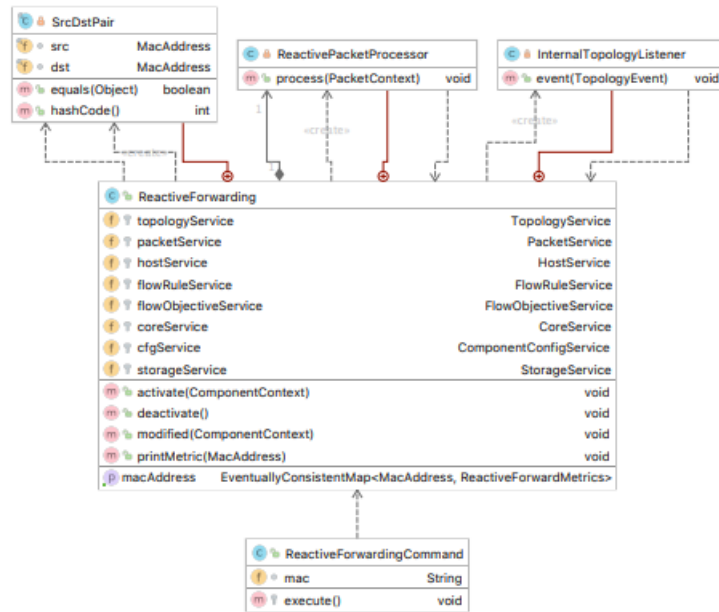


Figura 2.6: onos-fwd-app

#### 2.4.4 Intent Framework

O *intent framework* é um subsistema da ONOS que possibilita aos administradores de redes e aos aplicativos expressar as suas intenções de controle da rede em forma de política em vez da forma mecanizada, funciona como um *Northbound Interface* para as aplicações. O ONOS ao aceitar as especificações de intenções, as compila e instala. Ações estas que resultam em alterações na rede, como ligações a serem configuradas e novas regras de fluxos instaladas.

O *framework* do *intent* é organizado em uma estrutura de aplicação, o qual consiste num conjunto de funções projetadas para um determinado domínio de aplicação de modo que os desenvolvedores não precisam codificar do zero para construir uma app. Uma das suas melhores características consiste na habilidade de recompilar transparentemente um *intent* no caso de eventos na topologia, como falha nas ligações ou desconexão de dispositivos [9].

Embora o ONOS forneça um conjunto de *intents* incorporadas e os seus compiladores e instaladores, o framework do *intent* foi projetado para ser extensível. Isto permite a introdução de novas políticas e novos *intents*.

A figura 2.7 apresenta o fluxo compilação do *intent*. De início a app faz um pedido de instalação, se o pedido for aceite, o *intent* recebe um *intent* ID junto com o ID da aplicação atribuído pelo controlador. Logo o *framework* compila o *intent* recebido. No processo de compilação, um *intent* após ultrapassar a fase compilando, é traduzido num *intent* instalado para a rede. Contudo, a compilação pode apresentar dois resultados, sucesso ou compilação falhada. A compilação falhada coloca o *intent* no estado *Failed* o qual é um estado aonde ele pode ficar algum tempo. Caso contrário, se a compilação for um sucesso o *intent* será instalado.

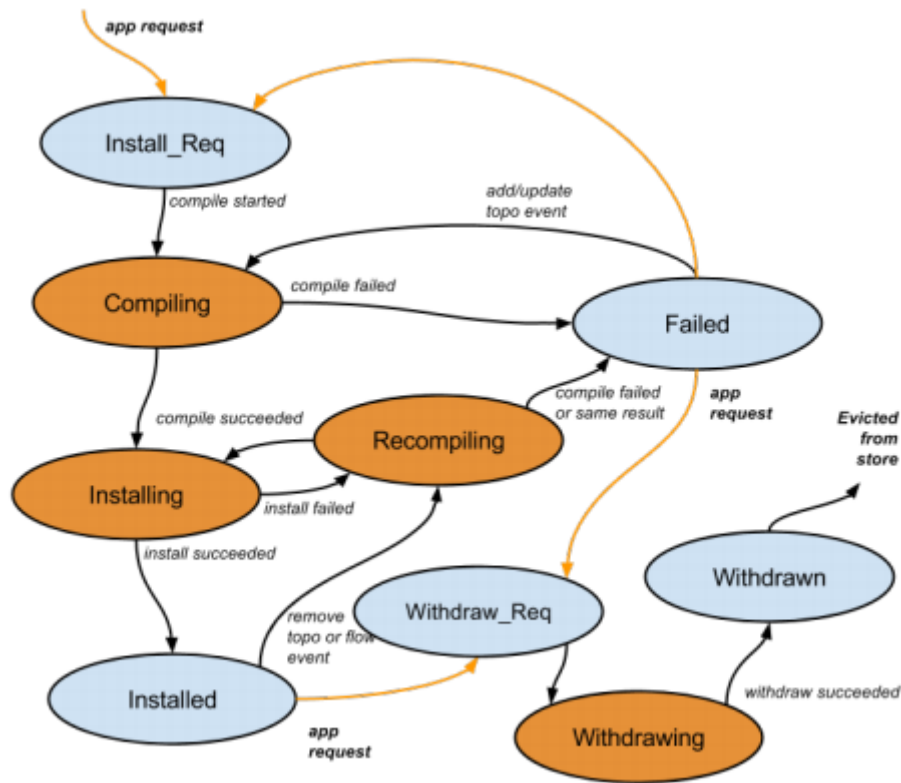


Figura 2.7: Compilação do *intent*

Igualmente a fase de compilação, o processo de instalação também pode falhar. No caso de ser um sucesso o *intent* é instalado, caso contrário o *intent* é recompilado. Se a recompilação também falhar o *intent* é colocado no estado *Failed*.

No estado instalado, um *intent* é traduzido e enviado as correspondentes especificações para os dispositivos na rede. Um *intent* já instalado também pode ser retirado com um *withdraw\_request*, nesse processo o controlador primeiro remove o *intent* e depois as entradas de fluxos configuradas nos dispositivos.

Os três principais casos de uso do *intent framework* são:

- **Criação de *intents*** - é quando um *intent* é criado e submetido, depois disso o *intent framework* irá compilar, armazenar e instalar regras de fluxos nos dispositivos.
- **Eventos da rede** - resulta de alterações no ambiente de rede como por exemplo falhas nas ligações, o sistema de monitoramento identifica a alteração e levanta um evento para recompilar o *intent*.
- **Retirada de *intents*** - é quando se faz um pedido de retirada do *intent*, o *intent framework* removerá o *intent* e as regras de fluxo instaladas nos dispositivos.

#### 2.4.4.1 Tipos de *intent*

O controlador ONOS traz por defeito a possibilidade de compilação de vários tipos de *intents*. A partir do seu CLI é possível a configuração dos seguintes:

- ***intent* Máquina-a-Máquina** - Instala a intenção de conectividade bidirecional entre dois hosts.
- ***intent* Ponto-a-Ponto** - Instala a intenção de conectividade unidirecional entre dois pontos da rede, isto é, entre duas portas dos dispositivos da rede.
- ***intent* Muitos-para-Um** - Instala a intenção de conectividade unidirecional entre múltiplas portas de entrada uma única porta de saída.
- ***intent* Um-para-Muitos** - Instala a intenção de conectividade unidirecional entre uma única porta de entrada e múltiplas portas de saída.

#### 2.4.4.2 Composição do *intent*

Um *intent* é composto por quatro grupos de elementos. Cada um destes elementos são constituídos por diversos campos que podem ser configurados pelo administrador de rede na submissão de uma política de *intent*.

- **Recursos de rede** - são conjuntos de objetos, como ligações e dispositivos que fazem parte da rede afetada pelo *intent*.
- **Critérios** - são campos ou padrões de cabeçalhos de pacotes que descrevem um tráfego. Os critérios são constituídos por vários campos como: Ethernet Type, endereço MAC de partida e chegada, endereço IP de partida e de chegada, protocolo IP, endereços Transmission Control Protocol (**TCP**) e User Datagram Protocol (**UDP**) e marcação DSCP.
- **Restrições** - são as limitações aplicadas a política de *intent* definido, como por exemplo a largura de banda e o tipo de ligação.
- **Instruções** - São as ações que devem ser aplicados ao tráfego selecionado, como por exemplo a alteração de um campo no cabeçalho do pacote e o reencaminhamento do pacote para uma outra porta de saída.

#### 2.4.5 Intent-based reactive forwarding

O *intent-based reactive forwarding* usa as intenções fornecidas para conectar os pontos de extremidade origem e destino enquanto fornece um caminho reativamente.

Na figura 2.8 um pacote para o destino host 2 é encaminhado para o comutador 1 pelo host 1. Como o novo pacote não possui nenhuma regra de fluxo o comutador 1 envia uma mensagem PKT\_IN, que contém o pacote incluído ao controlador para mais informações. O controlador processa o pacote e envia para o comutador 2 como uma mensagem PKT\_OUT que inclui ações como descartar ou encaminhar o pacote para uma porta específica. No processamento realizado pelo controlador, cria-se um *intent* baseado nos requisitos do pacote e envia-o ao *intent framework* para compilação e instalação. A compilação de um *intent* é um pedido assíncrono.

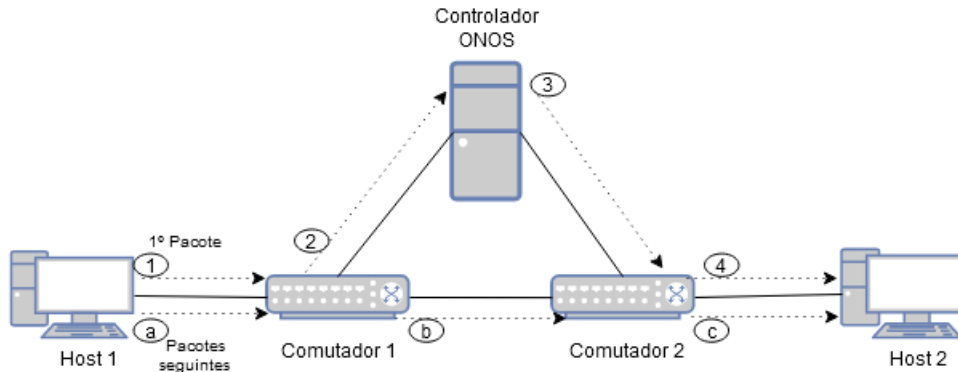


Figura 2.8: Intent Forwarding

#### 2.4.5.1 onos-ifwd-app

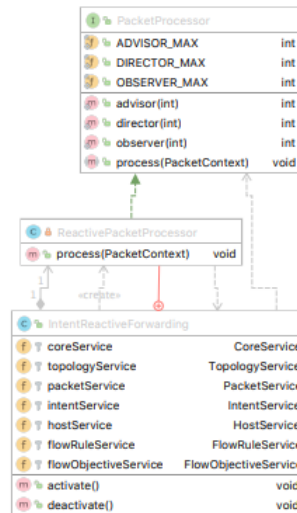


Figura 2.9: onos-ifwd-app

A app onos-ifwd-app representada na figura 2.9, é constituída pela classe pública *IntentReactiveForwarding* e pela classe privada *ReactivePacketProcessor*. Este segundo, é o responsável por verificar se o pacote já foi tratado anteriormente, caso sim, a app termina. Caso contrário, o pacote é analisado em parâmetros como por exemplo se o endereço de destino é conhecido. De seguida é configurado a conectividade instalando uma regra que encaminhe o pacote para o seu endereço de destino.

## 2.5 OpenFlow

Estabelecendo-se como a primeira interface de comunicação padrão entre o plano de controle e o plano de encaminhamento de dados, o OF permite o acesso direto e a manipulação do plano de dados de equipamentos de redes como comutadores e roteadores, tanto a nível físico como virtual. E trata-se de um papel muito importante pois move o controle de rede dos equipamentos (comutadores) para o *software* de controle logicamente centralizado (controlador) [15].

O protocolo OpenFlow (OF) é fundamental para a construção de redes SDNs. Concebido inicialmente em 2007 nas universidades de Stanford e da Califórnia, permite a manipulação e o acesso direto a dispositivos no plano de dados. Esta funcionalidade provém da utilização de tabelas de fluxos, comparando os pacotes recebidos com os valores na tabela de fluxo do dispositivo de rede associado, executando as ações correspondentes, conforme indicado na tabela. A figura 2.10 demonstra essa funcionalidade. Um fluxo é composto por um conjunto de pacotes, dentro de uma determinada janela de tempo, que possui valores iguais em alguns campos do cabeçalho. Possibilitando assim a criação de modelos de encaminhamento baseado na combinação de valores de cabeçalho definida pelo administrador de rede, o que se torna vantajoso para se ter um controle sobre as particularidades da rede.

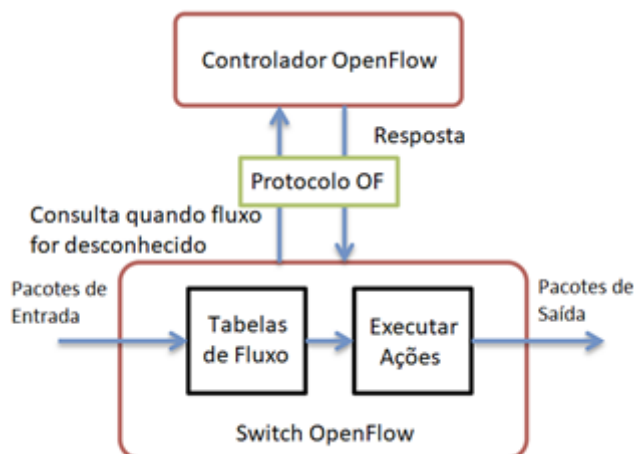


Figura 2.10: Funcionamento do protocolo OpenFlow

Utiliza-se o conceito de fluxos para identificar o tráfego de rede com base em regras de correspondência predefinidas que podem ser programadas estaticamente ou dinamicamente pelo *software* de controle SDN.

Uma vantagem do OpenFlow é permitir que o administrador de rede defina como o tráfego deve fluir no plano de dados tendo como base os padrões de uso, nas aplicações e nos recursos de nuvem [15]. Ao proporcionar a programação da rede através de fluxos, uma arquitetura baseada em SDN com OpenFlow, possibilita o controle granular da rede, permitindo que a rede seja adaptável as alterações em tempo real. As SDNs baseados em OpenFlow podem ser implantados em redes existentes, tanto físicas quanto virtuais. Dispositivos de rede podem suportar ser configuradas de maneira que algumas portas fazem o encaminhamento utilizando o protocolo

OpenFlow e outras não.

A Open Networking Foundation (ONF) é a responsável por padronizar o OpenFlow e o faz por meio de grupos de trabalho técnicos responsáveis pelo protocolo, configuração, teste de interoperabilidade e outras atividades, ajudando a garantir a interoperabilidade entre dispositivos de rede e *software* de controle de diferentes fornecedores. O OpenFlow está a ser amplamente adotado por fornecedores de infraestrutura, em geral através de uma simples atualização de *firmware* ou *software*.

O protocolo OpenFlow é um dos principais fatores que levaram a implementação da rede definida por *software*. Oferece aos desenvolvedores de aplicações de rede grande controle sobre a rede. Providenciando uma interface centralizada e acessível pela rede de modo a controlar o funcionamento dos dispositivos no plano de dados, enquanto, a lógica de controle dos dispositivos de rede como comutadores são centralizados num ou mais controladores. Permitindo aos administradores de redes fazer o gerenciamento dos dispositivos da rede de forma centralizada.

O OpenFlow consiste num conjunto de protocolos e uma API. Ele se baseia em encaminhamentos por fluxos, separando a rede em dois planos. O plano de controle é o responsável pelas configurações de controle de rede, enquanto, o plano de dados trata do encaminhamento do tráfego.

Os componentes principais da arquitetura estabelecida pelo OpenFlow são o controlador e o comutador com suporte a OF. Uma separação baseada nas redes definidas por *software*, como representada na figura 2.11.

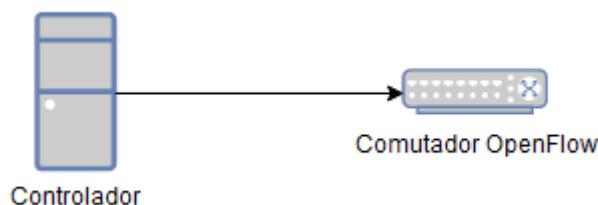


Figura 2.11: Arquitetura OpenFlow

Um comutador OpenFlow é constituído por três partes, o canal de comunicação seguro com o controlador, a interface do protocolo OpenFlow que permite ao controlador fazer o controle do comutador e a tabela de encaminhamento como na figura 2.12.

O canal de comunicação seguro com o controlador possui o protocolo SSL (Secure Socket Layer), adicionando maior proteção contra ataques e maior confiabilidade na troca de mensagens [24].

A interface do protocolo OpenFlow é a padronização das mensagens que chegam ao comutador provenientes do controlador de modo a definir comportamentos na rede. São instruções que permitem controlar de maneira genérica o plano de dados de qualquer dispositivo *hardware* que habilita o OpenFlow.

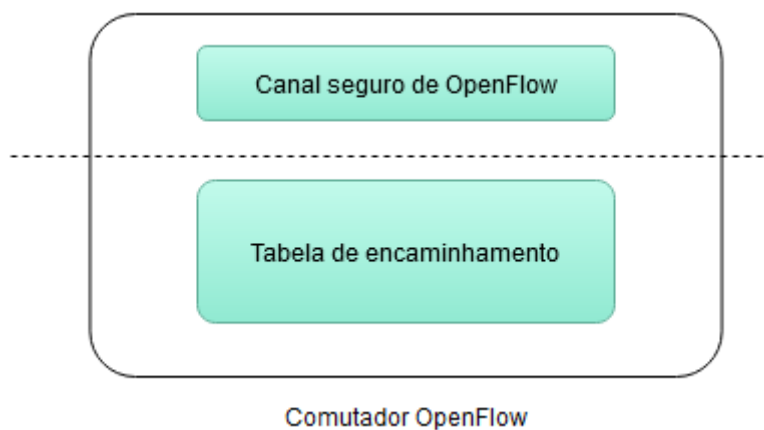


Figura 2.12: Comutador OpenFlow

A tabela de encaminhamento é constituída por regras de fluxos. Cada regra de fluxo consiste em um conjunto de instruções e ações associados ao tráfego na rede. A partir do uso da tabela de encaminhamento, o comutador faz o reencaminhamento de pacotes. A atualização da tabela fica ao cargo do controlador.

O cabeçalho Openflow é constituído por vários campos como apresentado na figura 2.13. São esses campos que são comparados com os campos dos pacotes que chegam no comutador, deste modo identificando o tráfego pretendido.

| Porta de entrada | Ethernet |         |      | VLAN |            | IP     |         |           |     | TCP/UDP |         |
|------------------|----------|---------|------|------|------------|--------|---------|-----------|-----|---------|---------|
|                  | Origem   | Destino | Tipo | ID   | Prioridade | Origem | Destino | Protocolo | ToS | Origem  | Destino |
| 1                | 2        | 3       | 4    | 5    | 6          | 7      | 8       | 9         | 10  | 11      | 12      |

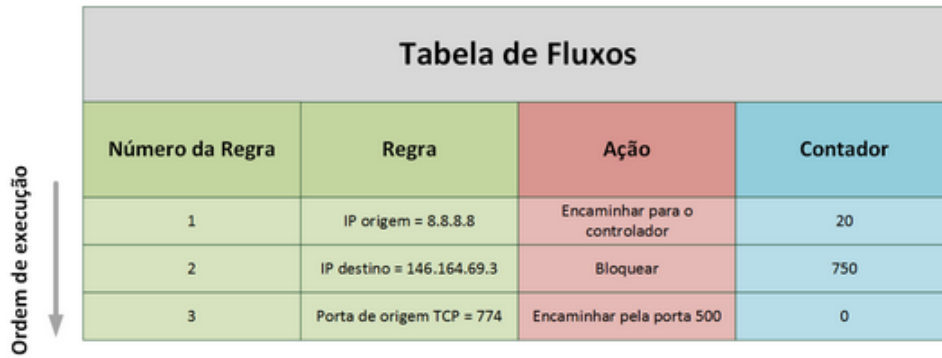
Figura 2.13: Cabeçalho OpenFlow[5]

Um fluxo é composto pela combinação de vários campos do cabeçalho de um ou mais pacotes de rede que passam pelos comutadores segundo o modelo TCP/IP. Os fluxos representam as características de um tráfego futuro. Podem representar tráfegos que ainda não passaram pelos comutadores mais que possuem as mesmas características, aos quais serão aplicadas as ações associadas ao fluxo.

Os fluxos encontram-se instaladas nas tabelas de encaminhamento ou de fluxo 2.14 dentro do comutador OpenFlow, deste modo o plano de dados executa as ações no tráfego pretendido.

As ações encontram-se associadas as regras de fluxos na tabela de encaminhamento, e são aplicadas aos tráfegos especificados pela regra associada. Ações como o encaminhamento, modificação de cabeçalho ou rejeição podem ser aplicados aos pacotes pertencentes a aquele fluxo.

A Open Networking Foundation (ONF) descreve todas as ações possíveis que podem ser aplicadas. A exemplo temos:



| Tabela de Fluxos |                           |                               |          |
|------------------|---------------------------|-------------------------------|----------|
| Número da Regra  | Regra                     | Ação                          | Contador |
| 1                | IP origem = 8.8.8.8       | Encaminhar para o controlador | 20       |
| 2                | IP destino = 146.164.69.3 | Bloquear                      | 750      |
| 3                | Porta de origem TCP = 774 | Encaminhar pela porta 500     | 0        |

Figura 2.14: Tabela de Fluxos

- Output-port - configura a porta de saída do tráfego;
- setQueue - configura a *queue* de saída o tráfego;
- setField - configura algum campo no cabeçalho do pacote;
- Push-Tag - coloca tag nos pacotes;
- Pop-Tag - retira tags dos pacotes;
- Drop - descarta o pacote;

O controlador é o responsável por instalar as regras de fluxos de tráfego na tabela de encaminhamento através da interface do protocolo OpenFlow. Representa o plano de controle na arquitetura OpenFlow, conectando-se de forma segura com o comutador OF.

A arquitetura do OpenFlow possibilita ao controlador ter uma vista e controle global da rede e recolher estatísticas sobre o estado da rede, que podem ser importantes nas decisões de configuração.

O controlador permite dois modos de comunicações possíveis com o comutador, como representada na figura 2.15. No modo **Out-of-band** é configurado um caminho específico só para a comunicação com o comutador OpenFlow, que não seja o caminho utilizado pelo OpenFlow para o tráfego de dados entre comutadores. Neste modo é necessário configurar VLANs ou portas que não estão submetidas ao OpenFlow.

No modo **In-band** a mesma ligação utilizada pelos comutadores para o tráfego OpenFlow é também utilizado pelo controlador para se comunicar com o comutador.

### 2.5.1 Mensagens OpenFlow

A troca de mensagens entre o controlador e os comutadores da rede ocorrem pelo canal seguro. Esta comunicação é iniciada pelo comutador, caso este saiba o endereço IP do controlador da rede. Cada mensagem OpenFlow entre o comutador e o controlador inicia com o cabeçalho OpenFlow,

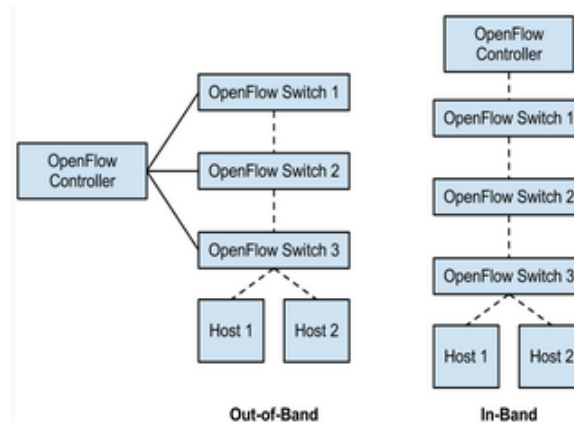


Figura 2.15: Comunicação In-band e Out-of-band[5]

contendo o número da versão do OpenFlow, o tipo da mensagem, o seu cumprimento e o ID da transição da mensagem [30]. As mensagens OpenFlow agrupam-se em três tipos diferentes, o controlador-para-comutador, o assíncrono e o simétrico descritos a seguir. Sendo que cada tipo possui os seus sub-tipos. As mensagens pertencentes ao sub-tipo Imutável, são mensagens que não vão ser alteradas em versão futuras do OpenFlow.

#### 2.5.1.1 Controlador-Para-Comutador

Este tipo de mensagens são iniciados pelo controlador e são usadas para fazer o gerenciamento ou inspecionar o estado dos comutadores. As mensagens Controlador-para-comutador estão divididas em 5 sub-tipos diferentes, que são: configuração do comutador, comando do controlador, estatísticas, configuração de filas e barreiras.

O sub-tipo configuração do comutador é constituído por uma mensagem de configuração unidirecional `SET_CONFIG`, o qual é utilizado pelo controlador para definir parâmetros no comutador. E por dois pares de mensagens pedidos-repostas, o `FEATURES` utilizado pelo controlador para interrogar o comutador sobre as configurações suportadas por ele e o `GET_CONFIG` utilizado para recuperar as configurações do comutador.

O sub-tipo comando do controlador é constituído por três mensagens. O `PORT_MOD` utilizado para mudar o estado de uma porta OpenFlow, o `FLOW_MOD` usado para modificar as regras de fluxos existentes na tabela de encaminhamento dos comutadores e o `PACKET_OUT` usado pelo controlador para enviar pacotes de dados para os comutadores para reencaminhamento no plano de dados.

O par de mensagens `STATS` constituem o sub-tipo estatísticas, utilizadas para obter os dados estatísticos dos comutadores. O sub-tipo barreiras é constituído pelo par de mensagens `BARRIER` utilizada pelo controlador para confirmar o término da execução de um certo comando OpenFlow enviado previamente. O comutador deve terminar a execução de todos os comandos recebidos anterior ao `BARRIER_REQUEST`, antes de executar qualquer outro comando recebido

Tabela 2.1: Mensagens OpenFlow [30]

| Mensagens                  | Tipo                       | Sub-tipo                  |
|----------------------------|----------------------------|---------------------------|
| PACKET_OUT                 | Controlador-para-comutador | Comando do controlador    |
| PORT_MOD                   | Controlador-para-comutador | Comando do controlador    |
| FLOW_MOD                   | Controlador-para-comutador | Comando do controlador    |
| BARRIER_REQUEST            | Controlador-para-comutador | Barreiras                 |
| BARRIER_REPLY              | Controlador-para-comutador | Barreiras                 |
| SET_CONFIG                 | Controlador-para-comutador | Configuração do comutador |
| GET_CONFIG_REQUEST         | Controlador-para-comutador | Configuração do comutador |
| GET_CONFIG_REPLY           | Controlador-para-comutador | Configuração do comutador |
| FEATURES_REQUEST           | Controlador-para-comutador | Configuração do comutador |
| FEATURES_REPLY             | Controlador-para-comutador | Configuração do comutador |
| STATUS_REQUEST             | Controlador-para-comutador | Estatísticas              |
| STATUS_REPLY               | Controlador-para-comutador | Estatísticas              |
| QUEUEUE_GET_CONFIG_REQUEST | Controlador-para-comutador | Configuração de filas     |
| QUEUEUE_GET_CONFIG_REPLY   | Controlador-para-comutador | Configuração de filas     |
| HELLO                      | Simétrico                  | Imutável                  |
| ECHO_REQUEST               | Simétrico                  | Imutável                  |
| ECHO_REPLY                 | Simétrico                  | Imutável                  |
| VENDOR                     | Simétrico                  | Imutável                  |
| PACKET_IN                  | Assíncrono                 | NA                        |
| FLOW_REMOVED               | Assíncrono                 | NA                        |
| PORTS_STATUS               | Assíncrono                 | NA                        |
| ERROR                      | Assíncrono                 | NA                        |

depois, devendo informar o controlador do término da execução destes comandos através do `BARRIER_REPLY`.

O sub-tipo configurações de filas é constituído pelo par de mensagens `QUEUEUE_GET_CONFIG_REQUEST` e `QUEUEUE_GET_CONFIG_REPLY`. Estes são utilizadas pelo controlador de modo a apreender como as filas estão configuradas no comutador. Desta forma o controlador pode mapear de maneira inteligente os fluxos de tráfego para determinadas filas de maneira a ter a qualidade de serviço pretendida.

### 2.5.1.2 Simétrico

As mensagens do tipo simétrico podem ser iniciadas tanto pelo controlador como pelo comutador. São constituídas por três mensagens, o `HELLO`, o `ECHO` e `VENDOR`. As mensagens `HELLO` são trocadas logo após o estabelecimento do canal seguro para a comunicação do controlador com o comutador, com o objetivo de determinar a versão mais alta do protocolo OpenFlow suportada tanto pelo controlador e pelo comutador. O OpenFlow especifica que a menor versão dentre os dois, deve ser a utilizada para a comunicação pelo canal seguro.

As mensagens `ECHO` são utilizadas em qualquer momento durante a ligação do canal seguro para confirmar que este ainda esteja ativo e também para avaliar a sua latência e largura de banda. As mensagens `VENDOR` estão disponíveis para melhoria ou experimentação específica de um fornecedor de comutadores.

### 2.5.1.3 Assíncrono

As mensagens assíncronas são enviadas a partir do comutador para o controlador, mesmo sem ter recebido nenhum pedido. É constituído pelas mensagens `PACKET_IN`, `ERROR`, `FLOW_REMOVED` e `PORT_STATUS`.

É através das mensagens `PACKET_IN` que os comutadores enviam os pacotes de tráfego de dados para controlador, em qualquer caso que seja necessário enviar um pacote ao controlador da rede. Quando uma regra de fluxos é removida da tabela de encaminhamento do comutador, este informa o controlador do ocorrido através da mensagem `FLOW_REMOVED`. As mensagens `PORT_STATUS` são utilizadas para informar o controlador sobre qualquer evento ocorrido no estado da porta, seja este causado pelo usuário diretamente ou alguma mudança física no comutador. As mensagens `ERROR` são utilizadas pelos comutadores para informar o controlador sobre a ocorrência de qualquer problema no funcionamento.

## 2.5.2 Falha na ligação comutador e controlador

A possibilidade de ocorrência de uma falha na ligação entre o controlador e o comutador é um aspeto já tratado pelo protocolo OpenFlow. A partir da sua versão 1.1 o protocolo já

específica o modo de funcionamento dos comutadores, caso estes perdem a ligação com todos os controladores [30]. No caso da perda de ligação com o controlador o comutador entra imediatamente no modo *fail secure mode* ou no modo *fail stand-alone mode*, dependendo de qual deles é suportado pelo comutador, caso os dois serem suportados, depende das configurações do administrador de rede.

No *fail secure mode* a única mudança no funcionamento do comutador são nas mensagens destinados ao controlador, estes serão descartados. No *fail stand-alone* o comutador cessa o funcionamento do OpenFlow e continua a funcionar na sua forma nativa como comutador ou roteador Ethernet [13].

No restabelecimento da conexão com o controlador, os fluxos previamente instalados nos comutadores continuam o seu funcionamento, contudo o controlador tem a opção de eliminar todas as regras de fluxos e configurar de novo a tabela de encaminhamento do comutador.

## 2.6 Mininet

O mininet é um emulador de rede de código aberto, o qual permite a criação de redes com comutadores virtuais, hospedeiros, ligações e controladores como representado na figura 2.16. Os host do mininet operam *software* de padrões da rede Linux, enquanto, os comutadores suportam OpenFlow, possibilitando a configuração de roteamento personalizado.

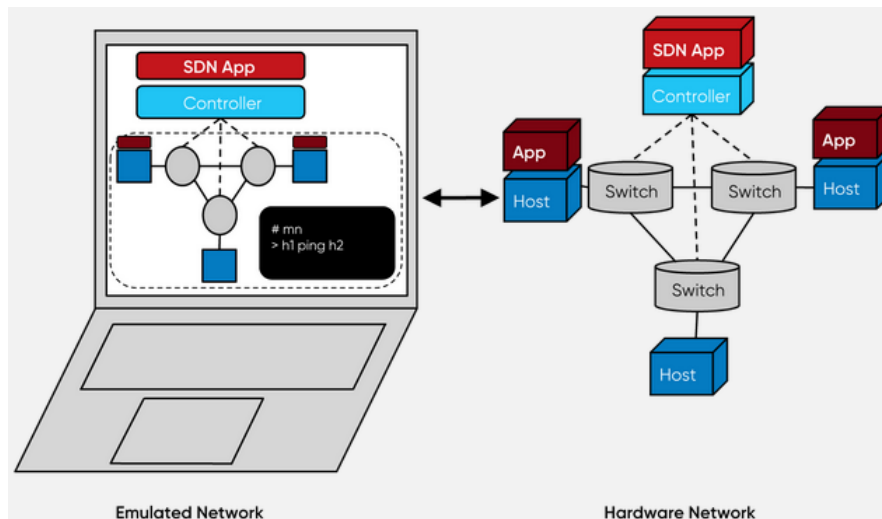


Figura 2.16: Visão geral do Mininet[11]

O mininet providência um *testbed* virtual e um ambiente de desenvolvimento para as redes definidas por *software* num PC ou *laptop* possibilitando assim várias configurações como, a criação rápida de um protótipo da rede SDN, a realização de testes de topologia sem a necessidade de ligar á uma rede física, a possibilidade de vários administradores de redes trabalharem de forma independente sobre a mesma topologia, o suporte a teste de regressão ao nível do sistema, a utilização do CLI para a depuração ou execução de testes em toda rede e um API Python simples

e extensível para a criação e experimentação da rede [23].

Uma rede mininet é constituída por [11]:

- **Hosts isolados** – um grupo de processos ao nível do utilizador, alocado a um *namespace* da rede, providência interface, portas e tabelas de encaminhamento exclusivo.
- **Ligações emulados** – as configurações de tráfegos de dados nas ligações são impostas pelo controlo de tráfego Linux onde cada host possui a sua própria interface virtual ethernet.
- **Comutadores virtuais** – A troca de pacotes através das interfaces é realizada pelo padrão Linux Bridge ou pela Open vSwitch executadas no modo Kernel. Estes podem ser utilizadas em modo kernel ou no espaço de usuário.

## 2.7 Open vSwitches

Os comutadores virtuais multicamadas Open vSwitch (OVS), representados na figura 2.17, são uma implementação de código aberto, com o objetivo de fornecer comutações para ambientes de virtualização de hardware, suportando diversos protocolos incluindo o OpenFlow. Ele é distribuído sob a licença Apache 2.0 [25].

O Open vSwitches (OVS) permite a automação de rede por meio de uma extensão programável, podendo ser feita a configuração automatizada da rede, demonstrando ser um ótimo recurso para ambientes virtualizados.

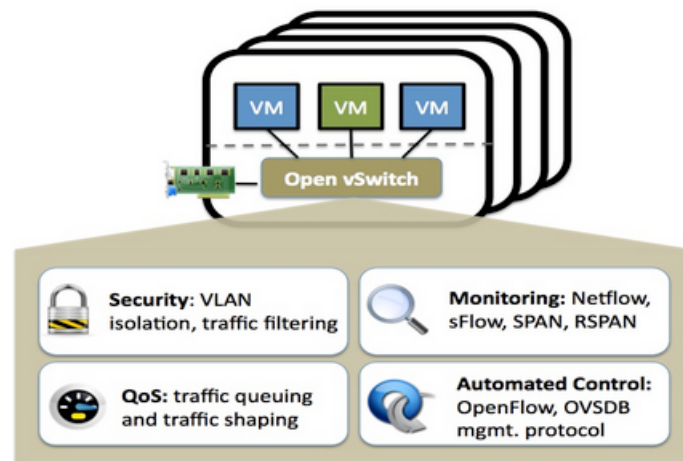


Figura 2.17: Open vSwitch[25]

O protocolo Open vSwitch Database Management Protocol (OVSDb), é utilizado nos controladores de modo a providenciar informações de configurações ao servidor de banco de dados dos comutadores, possibilitando assim a configuração de elementos internos do Open vSwitch.

O Open vSwitch é constituído por vários programas utilitários, onde cada um permite a configuração de uma das suas áreas. E entre os principais utilitários tem-se:

- **ovs-vsctl** – Configura o Open vSwitch.
- **ovs-ofctl** – Faz a administração das configurações OpenFlow no comutador Open vSwitch.
- **ovs-appctl** – Para comunicar com os módulos do Open vSwitch.
- **ovs-dpctl** – Permite configurar o encaminhamento de tráfego do comutador.

## 2.8 Topologias de Redes SDN

As seguintes topologias, são topologias permitidas pelo protocolo OpenFlow.

### 2.8.1 Topologia simples

Na figura 2.18, podemos observar uma topologia simples constituída por um controlador que gere a rede através de um comutador OpenFlow ligado a três hosts. O controlador tem o controlo total da rede.

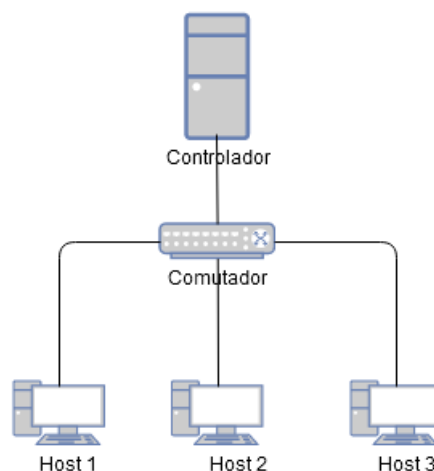


Figura 2.18: Topologia simples rede SDN

### 2.8.2 Um controlador para vários comutadores

Um controlador tem a capacidade de gerir vários dispositivos de rede. Como na figura 2.19 podemos ver uma topologia em que um único controlador faz o gerenciamento de quatro comutadores.

Esta abordagem possibilita ao controlador ter um vista global da rede e receber estatísticas de toda rede.

Esta topologia também permite ter comutadores conectados entre si como na figura 2.20, adicionando assim mais complexidade a rede. Esta foi a topologia utilizada na rede implementada

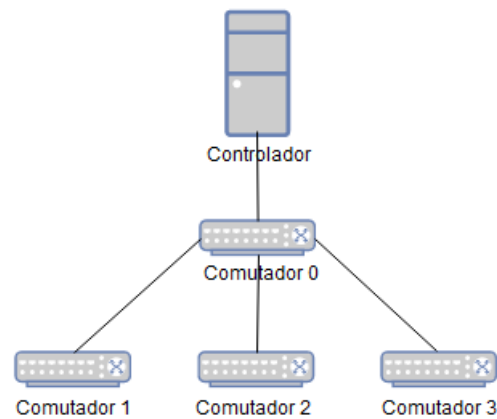


Figura 2.19: Um controlador para vários comutadores

durante a realização deste projeto.

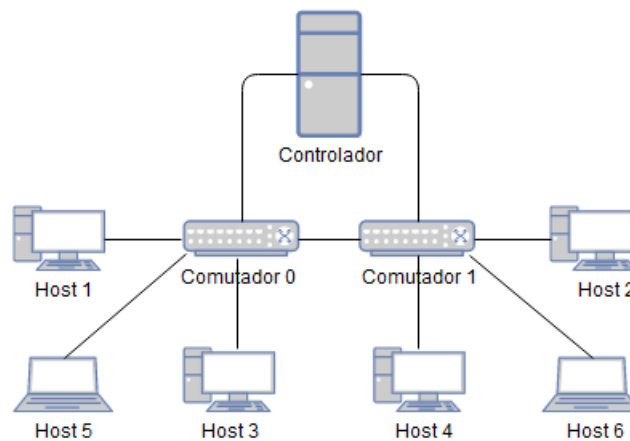


Figura 2.20: Um controlador para vários comutadores conectados entre si

### 2.8.3 Vários controladores para vários comutadores

Várias sub-redes podem estar ligadas entre si e ter o seu próprio controlador, como na figura 2.21. Nesta abordagem não há visão global da rede apenas por um único controlador. Para esta visão e controle global existir e ser compartilhada, os controladores deveriam trocar mensagens sobre a topologia de rede.

### 2.8.4 Controlador Distribuído

O protocolo OpenFlow também permite que uma arquitetura de rede distribuída seja aplicada ao controlador da rede, como na figura 2.22. Desta forma trouxe vantagens nas situações em que há um aumento considerável no tráfego, consequentemente aumentando o trabalho do controlador. Com a possibilidade de arquitetura distribuída, o controlador pode dividir a quantidade de

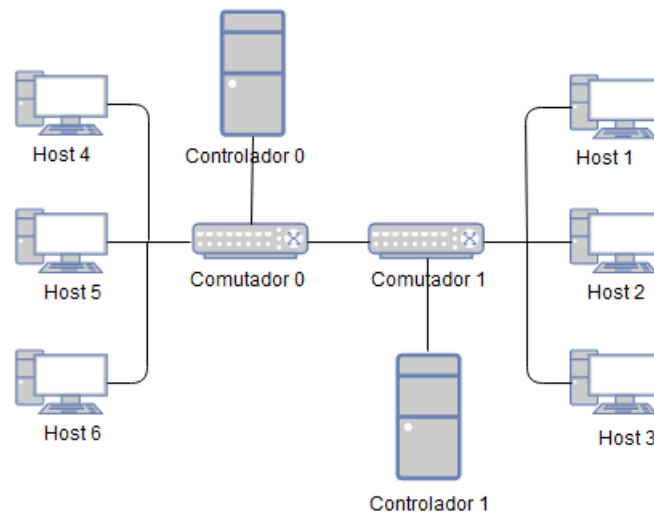


Figura 2.21: Vários controladores para vários comutadores conectados entre si

trabalho com outros controladores. Contudo mesmo usando uma arquitetura distribuída, esta é vista como sendo logicamente centralizada.

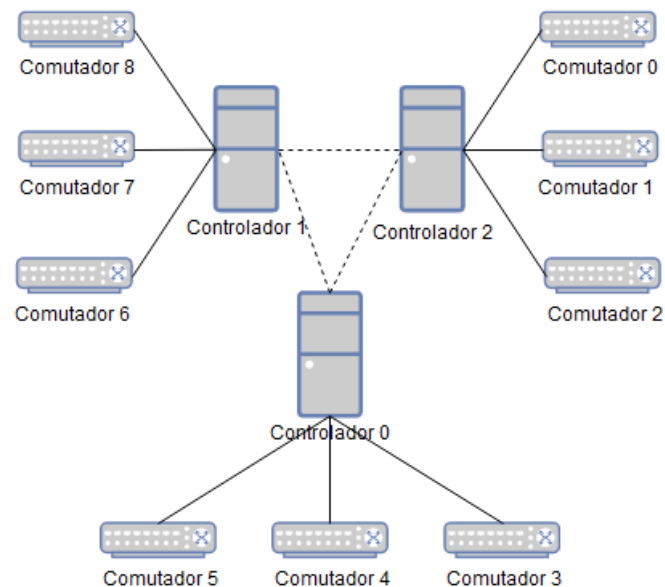


Figura 2.22: Controlador Distribuído

Numa arquitetura distribuída os vários servidores que compõem o controlador devem estabelecer um canal de comunicação aonde comunicam entre si, enviando mensagens de controle da rede de modo a fazer um controle global da rede baseando nas informações contidas em cada servidor.

O protocolo OpenFlow na sua versão 1.3 já permite que os comutadores sejam configurados de modo a funcionar com controladores distribuídos. Quando configuradas neste modo réplica, todos os servidores-controladores recebem as mesmas mensagens enviadas pelos comutadores.

Uma arquitetura distribuída traz mais segurança para a rede, tornando o mais confiável, uma vez que se um dos controladores falhar, os comutadores continuam a funcionar com as outras. Os diferentes controladores podem operar no modo Mestre/Escravo, aonde um deles é eleito mestre. Ao ser eleito um controlador mestre na rede, os comutadores alteram os outros controladores para modo escravo.

### 2.8.5 Arquitetura para Internet

Na figura 2.23, podemos observar uma estrutura de domínios ao longo da Internet. Nesta abordagem os comutadores internos aos domínios de rede tratam do encaminhamento do tráfego no interior do núcleo da rede. Os comutadores de borda tratam da ligação entre domínios, o qual pode ocorrer na camada de aplicação.

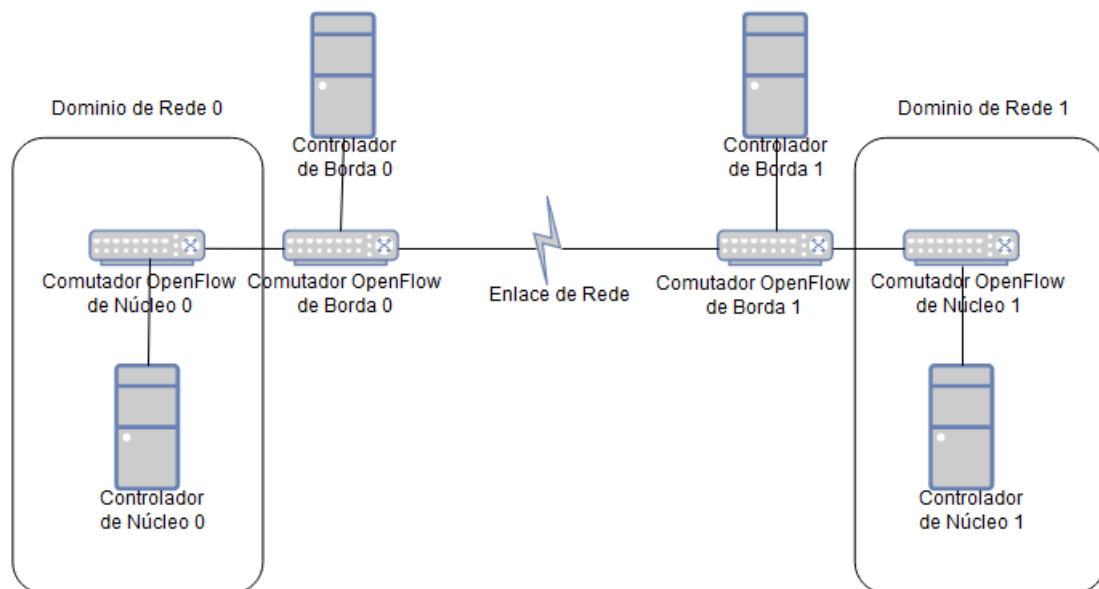


Figura 2.23: Arquitetura para Internet

### 2.8.6 Controladores centralizados e distribuídos

Numa rede SDN os controladores centralizados apresentam algumas desvantagens comparativamente ao modelo de controlador distribuído. O único controlador da rede representa um ponto único de falha, tornando-se um alvo prioritário para atacantes. Também é menos escalável comparado a arquitetura distribuída.

Contudo numa rede com controlador distribuído, este pode ser escalado para qualquer ambiente, e apresenta maior resiliência a falhas lógicas e físicas. No entanto pode apresentar uma certa dificuldade em manter atualizado todos os controladores sobre o estado da rede.

## 2.9 Tráfego multimédia

O ambiente de rede hoje em dia encontra-se dominado pelo tráfego de dados multimédia. Primeiramente, o usuário encontra-se cada vez mais acostumado ao acesso a banda larga, seja este em qualquer meio que esteja situado, o que conduz a um aumento exponencial do tráfego. O tráfego de multimédia refere-se a qualquer tipo de material, seja este áudio, vídeo, mensagens de voz, chamadas de vídeo etc., independentemente se este é distribuído a pedido ou em tempo real [26].

Com os avanços tecnológicos principalmente a nível de *hardware* que permitem a criação destes conteúdos, houve um aumento significativo dos conteúdos digitais disponível na rede para acesso, tornando assim os usuários grandes produtores assim como grandes consumidores destes conteúdos.

O acesso a informação multimédia é hoje um dos principais fatores da próxima geração de redes informáticas de comunicações. Além disso, estão a ser desenvolvidos cada vez mais novos produtos com o objetivo de alargar a capacidade de rede existente, de modo a dar resposta a essa demanda pelo tráfego de multimédia.

## 2.10 Voz sobre IP

Apesar da eficácia da rede existente atualmente e da sua capacidade de transporte das mais variadas medias, a transmissão de áudio profissional tem sido desafiador. Isto deve-se as várias características específicas do áudio no que diz respeito, a baixa latência exigida, a demanda por sincronização e a alta fidelidade especialmente nos ambientes produtivos.

Os engenheiros de áudio, são atualmente confrontados com enormes números de canais de áudios durante a realização de um evento. Como por exemplo a difusão de grandes performances de áudio ao vivo num estádio de futebol, requer a manipulação de até 2000 canais de áudio diferentes [6]. Outro exemplo seria num estúdio de gravação, aonde um grande número de ligações leva ao processo demorado de encaminhamento manual de sinais e atribuição de canais em relação a fonte.

As tecnologias de Voz sobre IP, podem ser usados com cabos ethernet, comutadores e roteadores, atenuando assim a necessidade de vários equipamentos especiais de som e a instalação de muitos cabos, exaltando uma das suas vantagens. No entanto, surgiu-se uma variedade de tecnologias de voz sobre IP, como por exemplo o Dante [4] e o CobraNet [8] que foram desenvolvidos e implantados.

Contudo elas eram interoperáveis entre si. Interoperabilidade que surgiu devido a diferentes formas de packetização de áudio, sincronização de tempo e protocolos de transporte. Contudo estes são amplamente utilizados em todo o mundo devidos a suas bem estabelecidas marcas no mercado e pelo suporte a transmissão de áudio multicanal sem compreensão em latências baixas

[6].

Em 2013 o Audio Engineering Society (AES) publicou o padrão AES67 “AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability” [1]. O objetivo desta publicação foi definir uma norma aberta de Voz sobre IP, muita próxima as tecnologias já existentes, de modo que os fabricantes estejam estimulados a desenvolver novas versões dos seus protocolos que sejam compatíveis com o padrão AES67, trazendo assim uma interoperabilidade entre as tecnologias de Voz sobre IP, como apresentada na figura 2.24.

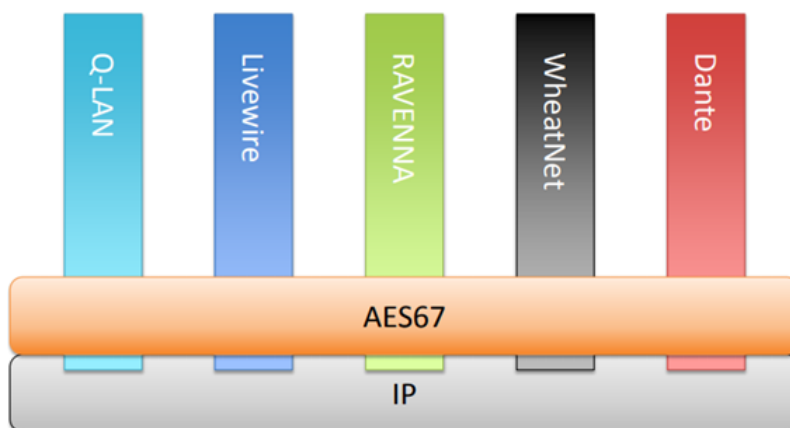


Figura 2.24: Interoperabilidade do padrão AES67

## 2.11 Principais diferenças entre a rede tradicional e a rede SDN

Devido a existência do *northbound interface* que permite ao controlador comunicar com as APIs, a rede SDN permite aos administradores de rede programar diretamente a rede, em vez de utilizar os protocolos exigidos numa rede tradicional.

Contudo a maior diferença entre as duas abordagens é que a rede SDN gera uma cópia abstrata da rede física, permitindo assim providenciar recursos a partir de um ponto centralizado. Ao contrário da rede tradicional, cuja localização física do plano de controle dificulta o gerenciamento da rede pelo administrador.

Deste modo, as redes SDNs providenciam aos administradores de redes, melhores soluções para fornecer recursos de acordo com as necessidades da rede. Encontram-se sintetizadas na tabela 2.2, algumas comparações mais específicas entre os dois modelos de rede.

## 2.12 Qualidade de Serviço

O Quality of Service (QoS) é descrito como um conjunto de meios que garantem o desempenho das aplicações, proporcionando-lhes meios e largura de banda suficiente para seu correto funcionamento. Baseia-se na ideia de que os tráfegos de redes não devem ser tratados de

Tabela 2.2: Rede SDN vs Rede Tradicional [28]

| <b>Critério</b>                                                           | <b>Rede tradicional</b>                                                                                                                 | <b>Rede SDN</b>                                                                       |
|---------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Gestão da rede                                                            | <ul style="list-style-type: none"> <li>• Difícil porque as configurações são implementadas separadamente em cada dispositivo</li> </ul> | <ul style="list-style-type: none"> <li>• Fácil com a ajuda do controlador;</li> </ul> |
| Visão global da rede                                                      | <ul style="list-style-type: none"> <li>• Difícil;</li> </ul>                                                                            | <ul style="list-style-type: none"> <li>• Visão central no controlador,</li> </ul>     |
| Custo de manutenção                                                       | <ul style="list-style-type: none"> <li>• Maior;</li> </ul>                                                                              | <ul style="list-style-type: none"> <li>• Menor;</li> </ul>                            |
| Tempo necessário para actualização configuração                           | <ul style="list-style-type: none"> <li>• Muito;</li> </ul>                                                                              | <ul style="list-style-type: none"> <li>• Pouco;</li> </ul>                            |
| Utilização do controlador                                                 | <ul style="list-style-type: none"> <li>• Não relevante;</li> </ul>                                                                      | <ul style="list-style-type: none"> <li>• Importante;</li> </ul>                       |
| Autenticidade, Integridade e consistência do controlador                  | <ul style="list-style-type: none"> <li>• Não relevante;</li> </ul>                                                                      | <ul style="list-style-type: none"> <li>• Importante;</li> </ul>                       |
| Integridade e consistência das tabelas de encaminhamento e estado da rede | <ul style="list-style-type: none"> <li>• Importante;</li> </ul>                                                                         | <ul style="list-style-type: none"> <li>• Importante;</li> </ul>                       |
| Disponibilidade do controlador                                            | <ul style="list-style-type: none"> <li>• Não relevante;</li> </ul>                                                                      | <ul style="list-style-type: none"> <li>• Importante;</li> </ul>                       |
| Utilização de recursos                                                    | <ul style="list-style-type: none"> <li>• Menor;</li> </ul>                                                                              | <ul style="list-style-type: none"> <li>• Maior;</li> </ul>                            |

forma igualitária, mas sim de forma específica possibilitando aos administradores de redes alterar a forma como os pacotes são tratados, como por exemplo, distinguir-lhes os tipos de serviços, ajustar a largura de banda, o atraso, a perda e criar filas virtuais de acordo com a sua específica prioridade.

No estudo do tráfego e análise do QoS tem-se principalmente os seguintes parâmetros:

- Atraso (latência)

Atraso ou latência é o atraso de tempo causado pela transmissão de pacotes de um ponto na rede para outro ponto. O delay na rede pode ser causado por vários motivos o que lhe leva a ter várias classificações como propagação do atraso, atraso de processamento, atraso na transmissão, entre outros [2].

- Tremor

Tremor é a variação do atraso. É afetado por variações no carregamento de tráfego e pela quantidade de colisões (congestionamento) entre os pacotes na rede. O efeito do jitter na performance da rede deve ser analisado em conjunto com o delay [2].

- Taxa de transferência

Taxa de transferência é a largura de banda atual, medida em um determinado tempo e em certas condições de rede que foram usadas para transferir pacotes de determinados tamanhos. O throughput da rede é soma das velocidades dos dados enviados para todos os dispositivos da rede [2].

O modelo da qualidade de serviço Integrated Service (**IntServ**), tinha sido proposto como um método que implementa a qualidade de serviço por salto, com reserva dinâmica de recursos para aplicações em tempo real. Basea-se no protocolo Resource ReServation Protocol (RSVP) para providenciar a qualidade necessária as aplicações. No entanto a maior desvantagem desta abordagem consiste na escalabilidade [16].

Contudo estas limitações levaram o Internet Engineering Task Force (**IETF**) a propor o modelo do Differentiated Service (**DiffServ**), com o objetivo principal de ultrapassar as limitações de escalabilidade experienciadas com o IntServ através da classificação de tráfego em Class of Service.

Posteriormente, surgiu um novo modelo bastante conhecido e utilizado, o Multi Protocol Label Switching (**MPLS**). Este basea-se na marcação dos IPs com rótulos indicando o menor caminho entre dois hosts, deste modo evitando o processo de procuras complexas nas tabelas de encaminhamento [16].

### 2.12.1 DiffServ

O Differentiated Service (**DiffServ**) é um método que permite a classificação e o tratamento diferenciado do tráfego de dados, providenciando assim a qualidade de serviço numa rede de IPs. A arquitetura do DiffServ especifica o uso do campo Differentiated Services (**DS**) *field* nos cabeçalhos de pacotes IPv4 e IPv6 para transportar um dos 64 valores de *Differentiated Services Code Point* (**DSCP**).

O DiffServ é um método de gestão de tráfego baseado em classes e de granularidade grande. Este depende de um mecanismo para classificar e marcar os pacotes pertencentes a diferentes classes. Os dispositivos que suportam o DiffServ implementam um comportamento por salto no encaminhamento de pacotes associados a uma determinada classe. Podem ser definidos diferentes níveis de comportamento por salto, oferecendo assim um tratamento diferente para cada classe de dados.

O comportamento por salto Per-hop behavior (**PHB**) é fundamental para o funcionamento do DiffServ. Em cada dispositivo da rede, um determinado tráfego é transmitido de acordo com

a sua específica classificação. O DiffServ não diferencia os fluxos de tráfego individualmente, mas opera com base na classificação do tráfego, colocando cada pacote num número limitado de classes de tráfego. Deste modo cada classe de tráfego pode ser gerida de forma diferente.

#### 2.12.1.1 Classificação

O tráfego que chega num dispositivo que suporta o DiffServ esta sujeito a uma classificação e um condicionamento. A classificação de tráfego ocorre de acordo com o conteúdo do cabeçalho do mesmo, este pode já trazer um determinado valor DSCP e prosseguir com o mesmo DSCP, ou então pode também ser remarcado com um novo DSCP, recebendo assim uma nova classificação de menor ou maior prioridade para o próximo salto na rede.

Os 64 diferentes valores de DSCP encontram-se mapeados nas seguintes classes de tráfego:

- **Default Forwarding (DF)** – Esta é a única classe de tráfego obrigatório. Qualquer tráfego que não satisfaz os requisitos para as outras classes utiliza esta classificação. O valor DSCP para esta classe é 0.
- **Expedited Forwarding (EF)** – Esta classificação é caracterizado por apresentar baixo tempo de atraso, baixo jitter e baixa perda. A taxa de erros é controlada e a largura de banda associada ao tráfego é garantida.
- **Assured Forwarding (AF)** - Esta classificação permite ao administrador garantir a entrega do tráfego, desde que este não exceda alguma condição, como por exemplo a largura de banda. Caso isso acontecer tem uma maior probabilidade deste ser perdido se ocorrer um congestionamento.

#### 2.12.2 AES67

O AES67 é um padrão técnico para se obter a interoperabilidade na transmissão de voz sobre IP. O padrão foca em definir, como os protocolos existentes são usados para se obter a interoperabilidade em redes de 3 camadas [1]. Providenciando assim recomendações específicas para se atingir a interoperabilidade entre diferentes tecnologias de voz.

A capacidade de sincronização e de partilha de um relógio em comum por todos os participantes da rede constitui uma parte muito importante na transmissão de voz sobre IP. A diferença entre uma transmissão com ótima performance em relação a uma de baixa performance, muitas vezes encontra-se na má sincronização entre os dispositivos da rede.

Um relógio comum permite que todos os fluxos de amostras de áudio são encapsulados e recebidos ao mesmo tempo, possibilitando assim uma latência fixa e determinável entre o emissor e o recetor.

Esta configuração é fundamental para se conseguir uma transmissão eficiente de áudio e deve ser efectuada utilizando o Protocolo de Precisão de Tempo (PTPv2) IEEE 1588-2008.

Os dados multimédia devem ser transportados utilizando o protocolo IPv4. Uma vez empacotados, são transportados com base no modelo OSI. O AES67 define parâmetros nas camadas 3 e 4, mas não especifica como a interoperabilidade é conseguida nas outras camadas abaixo.

A qualidade de serviço constitui uma parte fundamental na transmissão de voz sobre IP. Numa rede partilhada o tráfego de áudio exige um tratamento personalizado em relação aos outros tipos de tráfego, devido a sua alta sensibilidade em relação ao delay e jitter. Deste modo para se obter uma ótima performance na transmissão de áudio sobre IP, todos os dispositivos na rede devem ser configurados de maneira a ter um comportamento semelhante em todo o caminho de dados e minimizando da melhor forma possível o tempo de espera.

De forma a conseguir realizar um tratamento adequada para a Qualidade de Serviço na rede, os dispositivos devem implementar o método do Differentiated Service (DiffServ). Este utiliza o campo DSCP no cabeçalho dos pacotes IP para fazer marcações de acordo com o tráfego a ser tratado no momento, permitindo assim facilmente distinguir os pacotes que merecem ser tratados preferencialmente.

O AES define três tipos de classes e os seus respetivos valores DSCP recomendados que devem ser utilizados na priorização da transmissão de voz sobre IP, valores apresentados na tabela 2.3. Dispositivos devem marcar tráfegos de saída com o seu respetivo valor DSCP. Todos os dispositivos da rede devem usar os valores propostos pelo AES67, contudo um administrador de rede pode também definir tráfegos priorizados com outros valores DSCP.

Tabela 2.3: Classes de QoS e DiffServ

| Nome da Classe     | Tipo de tráfego                                                                                                                                                       | Default DiffServ Class (valor decimal DSCP)                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| <b>Clock</b>       | <ul style="list-style-type: none"> <li>IEEE 1588-2008 Announce, Sync, FollowUp, DelayReq, DelayResp, PdelayReq, PdelayResp and PdelayRespFollowUp packets;</li> </ul> | <ul style="list-style-type: none"> <li>EF(46);</li> </ul>   |
| <b>Media</b>       | <ul style="list-style-type: none"> <li>RTP and RTCP media stream data;</li> </ul>                                                                                     | <ul style="list-style-type: none"> <li>AF41(34);</li> </ul> |
| <b>Best effort</b> | <ul style="list-style-type: none"> <li>IEEE 1588-2008 signaling and management messages. Discovery and connection management messages;</li> </ul>                     | <ul style="list-style-type: none"> <li>DF(0);</li> </ul>    |

Com a publicação do padrão AES67, os produtores das tecnologias de Voz sobre IP têm agora forte incentivo em desenvolver novas tecnologias interoperáveis através do AES67, o que

seria uma grande vantagem para a produção e distribuição de áudio.

O AES67, promove a interoperabilidade e a formação de maiores redes de transmissão de voz, no entanto, este não contém uma solução para o problema de gerenciamento e configuração da rede. Uma vez que a sua função consiste na interoperabilidade da transmissão de amostras de áudio, este não aborda os desafios de roteamento de fluxos na rede nem a descoberta e configuração de novos dispositivos.

Numa implementação tradicional, um engenheiro de áudio tem de decidir, se uma determinada rede é ou não adequada para uma transmissão de voz, baseado na topologia da rede, na latência e na sua taxa de transmissão [6]. O que implica muito trabalho manual, desde o adicionamento de novos cabos ethernet, comutadores, roteadores e a configuração manual de cada um desses dispositivos. Tornando assim uma tarefa cara e exaustiva, elevando a probabilidade de erro.

- **Clock**

O tráfego de relógio (PTP) consiste em transmissões de baixas frequências de pequenos pacotes UDP com largura de banda baixa. Embora o tráfego de relógio não é muito sensível a perda de pacotes, este é sensível a latência e especialmente sensível ao jitter. Deste modo o AES recomenda que seja atribuída maior prioridade ao tráfego de relógio IEEE 1588-2008. Na maioria das redes esta prioridade é conseguida atribuindo ao tráfego de relógio a classificação de *Expedited Forwarding* (EF)(46) PHB. A classe EF é implementada com uma fila de prioridade estrita, o qual tem maior prioridade de transmissão em relação às outras filas. Para uma maior consistência na priorização do tráfego de relógio, este pode ainda ser reconhecido e classificado com base no seu endereçamento IP e número de porta.

- **Media**

O tráfego media (RTP e RTCP) é caracterizado por alta frequência e alta largura de banda. O tamanho dos pacotes depende do número de canais de áudio transportados num fluxo. O tráfego de pacotes de mídia deve ser entregue de forma atempada e fiável. Qualquer perda de pacotes vai manifestar-se na queda de áudio. Este impacto varia de aplicação para aplicação, mas a maioria tem baixa tolerância para perdas.

O AES recomenda como classificação ideal para o tráfego de media a maior prioridade da classe AF, ou seja, o valor DSCP AF41(34), mas também permite utilizar outras marcações AF para diferentes fluxos de tráfego. O melhor desempenho é alcançado se o AF for implementado como uma segunda fila de prioridade rigorosa abaixo da EF. A implementação do AF pode exigir que o administrador de rede estabeleça uma taxa de subscrição que suporta o tráfego que se espera que seja transportado pela classe.

Do mesmo modo, para se obter maior consistência na sua priorização, este pode ser classificado com base no seu endereçamento IP e porta.

- **Best effort - BE**

O Best-effort tráfego consiste em qualquer tráfego relacionado com o AES67 que não seja classificado como tráfego de relógio ou de media, como por exemplo, o envio de mensagens associadas a descoberta e gestão de ligações.

O tráfego best-effort normalmente é marcado com o valor DSCP de BE(0). Porque todas as redes implementam um tráfego best-effort, qualquer tráfego não classificado de outra forma é considerado como um tráfego best-effort. Normalmente não é necessária nenhuma configuração especial para acomodar o tráfego BE.

### 2.12.3 Queues

Na arquitetura de redes de computadores uma *queue* ou fila é utilizada para armazenar tráfego nos comutadores ou roteadores até este poder ser processado ou serializado. Todas as interfaces dos comutadores e roteadores possuem filas de entrada (*inbound*) e filas de saída (*outbound*). As filas de entrada armazenam pacotes até o CPU do comutador ou roteador poder reencaminhar o pacote para a interface correta, enquanto, as filas de saída armazenam pacotes até o comutador ou roteador poder serializar o pacote para a ligação física.

A congestão das filas ocorre quando a taxa do tráfego de entrada é maior do que o tráfego que pode ser processado corretamente pela fila de saída. Esta situação pode ocorrer nos seguintes momentos, quando a velocidade de interface de entrada é maior do que a interface de saída, quando o tráfego combinado de várias interfaces de entrada excede a capacidade da interface de saída e quando a capacidade do comutador ou roteador é insuficiente para suportar o tamanho da tabela de encaminhamento.

No seu funcionamento padrão se o buffer de uma fila ultrapassar a sua capacidade, os novos pacotes que chegam a fila são descartados independentemente da sua classificação ou marcação.

Contudo a **QoS** proporciona comutadores e roteadores com um mecanismo de filas e serviços que processam tráfegos de maiores prioridades antes dos tráfegos de menor prioridade. Isto é, atribuindo a cada fila prioridade e largura de banda diferentes. Juntamente com um sistema de *drop* que permite descartar tráfego com menor prioridade antes do tráfego com maior prioridade nos momentos de congestão.

Este tratamento diferenciado pode ser feito baseado em *Class of Service (CoS)*, e usando marcações *DSCP* para marcar os pacotes com menos ou mais prioridade [32].

### 2.12.4 As vantagens da rede SDN para tráfego de voz sobre IP

A rede SDN consiste num grande avanço para a aperfeiçoamento do tráfego de voz sobre IP, pois este traz muitas vantagens, desde a configuração da rede até o seu gerenciamento e qualidade

de serviço. A sua flexibilidade juntamente com a capacidade de automatização da rede permite que várias configurações sejam feitas dinamicamente com base em eventos que ocorrem na rede, constituindo assim um ativo ideal para atender aos requisitos do tráfego de voz.

Dentre as vantagens providenciadas pela rede SDN temos o gerenciamento e configuração da rede. Este permite ter uma visão geral e central da topologia da rede, possibilitando ao engenheiro de áudio um melhor planejamento e roteamento dos fluxos por toda a rede. Outra vantagem providenciada pela rede SDN é o descobrimento dinâmico de novos dispositivos conectados na rede. Libertando o engenheiro de áudio de todo o trabalho manual que era preciso numa rede tradicional e diminuindo assim a probabilidade de erros durante a configuração.

Contudo a maior vantagem da rede SDN para o tráfego de voz sobre IP é a sua forma inteligente e eficiente na implementação da qualidade de serviço [29]. A QoS é fundamental em qualquer implementação do tráfego de voz sobre IP, este tem de ser aplicado em todos os dispositivos que fazem parte do caminho do tráfego de voz e tradicionalmente a QoS é configurado individualmente em cada dispositivo. Deste modo o SDN permite ultrapassar os seguintes desafios:

- Detecção e localização de falhas particulares na rede, indicando-lhes ao administrador da rede ou ajustando-os dinamicamente;
- Permite que várias configurações sejam aplicadas simultaneamente na rede, diminuindo assim o tempo de inatividade;
- Interpretar requisitos de QoS e traduzi-las em configurações apropriadas que são automaticamente enviados em forma de fluxos para os dispositivos e aplicados;
- Permite a harmonização de todas as configurações QoS em todos os dispositivos da rede, garantido assim um QoS ponta a ponta.

## Capítulo 3

# Ambiente de teste SDN

Este capítulo descreve os passos executados e as ferramentas utilizadas para a implementação da rede SDN.

A implementação do ambiente de rede definida por *software* caracterizou-se por 3 fases principais. Inicialmente com a configuração de um controlador, o segundo foi a configuração dos três comutadores HP Aruba e a terceira foi a configuração do protocolo OpenFlow e a consequente ligação entre o controlador e os comutadores.

### 3.1 Configuração do controlador

Na implementação do ambiente SDN, o primeiro passo foi configurar uma máquina para operar como controlador da rede. Deste modo configurou-se a máquina com o sistema operativo Linux Ubuntu 20.04.1 LTS, e com 6GB de memória.

À máquina do controlador foram adicionadas duas placas de rede, uma para comunicação com os comutadores na rede SDN, a outra para conexão a rede Internet externa. A primeira interface, para a rede SDN foi configurada com o endereço IP estático 192.168.100.3 e máscara 255.255.255.0. Deste modo todos os outros dispositivos da rede foram configurados dentro da mesma gama de endereço.

A escolha do OpenFlow controller ONOS, fez-se, pelo facto de este possuir uma máquina virtual associada com o Mininet [11], possibilitando assim um melhor estudo e conhecimento da mesma, projetando uma melhor experiência mesmo antes da sua real implementação no controlador.

O controlador ONOS foi criado com o objetivo de oferecer mais flexibilidade para criação e implantação de novos serviços de rede dinâmicas com interfaces programáticas e simplificadas.

O ONOS suporta configuração e controle em tempo real da rede. Ao mover a inteligência para o controlador ONOS, os usuários podem criar facilmente aplicativos de rede sem a necessidade

de alterar os sistemas de plano de dados [12]. A plataforma ONOS inclui:

- Uma plataforma e um conjunto de aplicações que atuam como um controlador SDN modular, distribuído e extensível.
- Gestão, configuração e implantação simplificados de novos *softwares*, *hardwares* e serviços.
- Uma arquitetura escalável para fornecer a resiliência e a escalabilidade necessárias para atender aos rigores dos ambientes SDN.

Após a escolha do *software* ONOS, foi efetuada a instalação seguindo as instruções na página oficial [12]. Contudo nesta fase houve alguma dificuldade em configurar o *software* de maneira que identificasse a interface correta para os comutadores SDN, uma vez que a máquina tem duas interfaces de rede. O controlador escolhe a interface de rede principal e conecta-se automaticamente. Contudo este problema foi ultrapassado ao iniciar o serviço do *software* ONOS com apenas uma interface ligada no momento da inicialização de serviço.

Desta forma com o serviço do ONOS iniciado, a máquina já funciona como um controlador de rede SDN, já permitindo o acesso ao seu CLI, Graphical User Interface (GUI) e API.

### 3.2 Configuração comutadores HP Aruba

Os comutadores utilizados na realização deste projeto foram os modelos da Hewlett packard (HP) Aruba 2930M 24G PoE JL320A. Estes possuem o HPE Smart Rate e portas multi-gigabit que permitem alta velocidade para pontos de acessos e dispositivos IoT [19].

Os comutadores HP Aruba 2930M favorecem a implementação da rede SDN, suportando interfaces programáticas incluindo o REST API e o OpenFlow na sua versão 1.0 e 1.3, os quais permitem a automação das operações de rede assim como monitoramento e resolução de problemas.

Estes também possuem várias outras características que permitem configurar as necessidades da qualidade de serviço da rede. Como por exemplo, a priorização do tráfego de tempo real com níveis de prioridade mapeadas em oito filas de prioridade, a priorização nível 4 baseada em TCP/UDP, o *Class of Service (CoS)* baseado no endereço IP, no *Type of Service (ToS)*, no protocolo de nível 3, na porta TCP/UDP e no *Differentiated Service*.

Com base na documentação de instalação do comutador HP Aruba 2930M publicada pela aruba a Hewlett Packard Enterprise Company procedeu então a sua configuração [20]. Utilizando a porta de ligação para consola fez-se a configuração do endereço IP e máscara para as suas portas de tráfego. Com essas configurações já foi possível aceder ao comutador através da máquina que

contém o controlador, com um cabo RJ45 ligado a uma das suas portas de tráfego, utilizando ferramentas de acesso <sup>1</sup>.

Contudo apesar de já aceder ao comutador a partir da máquina que funciona como controlador de rede contendo o ONOS, este não identifica o comutador como sendo um dispositivo da rede SDN, pois ainda não foi habilitado o protocolo OpenFlow no comutador HP.

### 3.3 Configuração OpenFlow

Para efectuar a configuração do protocolo OpenFlow (**OF**) no comutador HP Aruba. Não foi necessário a instalação de nenhum outro protocolo ou *software*, uma vez que o comutador suporta o OpenFlow, possibilitando a sua configuração através do CLI.

Acedendo ao CLI do comutador HP a partir da máquina que contém o controlador utilizando o ssh, uma vez que o comutador já tinha sido configurado com um endereço IP estático, fez-se a habilitação do protocolo OpenFlow com os seguintes passos:

Primeiramente foi ativado protocolo OpenFlow. De seguida configurou-se um controlador OpenFlow para o comutador, utilizando um ID particular e atribuindo-lhe o IP 192.168.100.3 e a porta 6633, pois é o endereço do controlador ONOS. Também se configurou a interface de comunicação utilizada para comunicar com o controlador. Neste caso como o comutador HP Aruba possui uma porta OOBM, esta foi utilizada para ser o canal de ligação com o comutador. Contudo podia ser usada qualquer outra porta, no entanto, esta deveria estar numa Virtual Local Area Network (**VLAN**) especialmente configurada para a comunicação com o controlador, separado das outras portas de tráfego.

O próximo passo então foi a configuração da porta OOBM do comutador HP com um endereço IP e uma máscara de rede, dentro da mesma gama de endereços que o controlador.

O passo seguinte foi configurar as instâncias OpenFlow no comutador. A instância OpenFlow é uma configuração muito interessante do comutador HP, pois ele permite configurar que algumas portas de tráfego do comutador operam com o protocolo OpenFlow, enquanto, outras não. Possibilita a configuração de várias instâncias, cada uma com a sua VLAN. Contudo, o comutador HP também possibilita a configuração da instância no modo *aggregate*, no qual todas as portas de tráfego do comutador operam com o protocolo OpenFlow. Para esta implementação foi configurada o modo *aggregate* para a instância OpenFlow criada no comutador HP e foi utilizada de início a versão 1.0 do OpenFlow.

Desta forma o comutador já se encontra configurado para integrar a rede SDN e se conectar ao controlador. Contudo é de salientar que mesmo tendo essas configurações realizadas, para o controlador ONOS identificar o comutador como sendo um comutador OpenFlow conectado a sua rede, foi preciso a ativação da aplicação *OpenFlow Base Provider* nas apps do ONOS, o que

---

<sup>1</sup>putty - <https://www.putty.org/>

levou a ativação automática das seguintes apps, *Host Location Provider*, *LLDP Link Provider*, *OpenFlow Base Provider*, *Optical Network Model*.

Após a configuração bem sucedida do primeiro comutador HP, realizou-se as mesmas configurações nos outros dois comutadores HP disponíveis.

Na figura 3.1, é possível observar o GUI do controlador ONOS após as configurações realizadas. Pode-se constatar que o ONOS já identifica na sua topologia os três comutadores HP Aruba configuradas.

No appendix, encontra-se disponíveis todas as configurações específicas realizadas, em relação ao endereço IP e máscara atribuída a cada comutador e as suas portas OOBM.

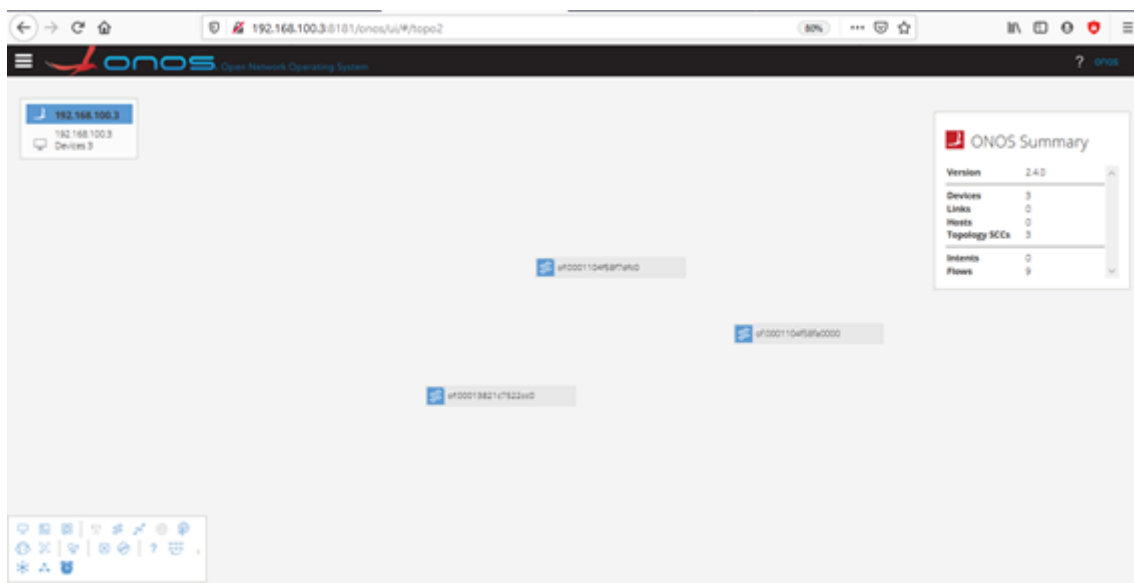


Figura 3.1: ONOS GUI

Na figura 3.2 podemos observar a topologia da rede implementada, aonde as ligações a vermelho representam as ligações OpenFlow entre os comutadores e o controlador, e as ligações a preto são os caminhos de tráfego OpenFlow. Nesta implementação foi utilizada um comutador simples, sem suporte a OpenFlow de modo a estabelecer uma ligação *out-of-band* entre o controlador e cada um dos 3 comutadores HP Aruba, uma vez que a máquina com o controlador só possui uma interface de rede.

Inicialmente os comutadores HP Aruba foram configurados com o OpenFlow na sua versão 1.0. Contudo após alguns testes de funcionamento demonstrados no capítulo 4 desta dissertação, procedeu-se então a alteração do protocolo OpenFlow da versão 1.0 para versão 1.3.

A principal diferença entre as duas versões está nas tabelas de fluxos. Na versão 1.0 o OpenFlow tem o conceito de apenas uma tabela de encaminhamento nos comutadores, onde são armazenadas todas as regras de fluxo, por outro lado, na versão 1.3 há a possibilidade de ter múltiplas tabelas de encaminhamento. Este novo conceito de múltiplas tabelas permite a criação de uma hierarquia de fluxos usando *pipeline*, isto é, permite criar regras numa determinada

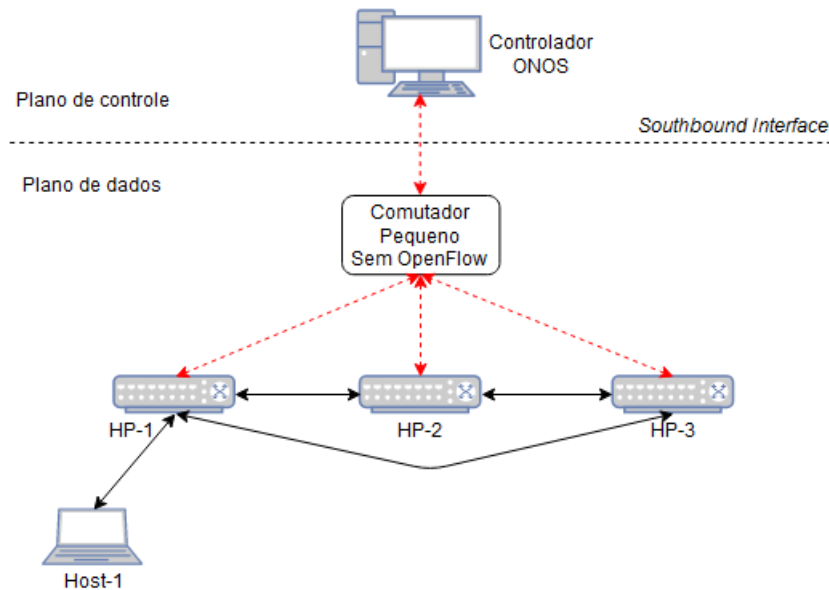


Figura 3.2: Topologia da rede implementada

tabela de fluxo em que a ação associada a esta regra diz ao controlador para enviar aquele fluxo para uma outra tabela, e assim sucessivamente.

Outra vantagem da versão 1.3 está nos campos de especificações para a correspondência com o tráfego, este possui mais possibilidade de configuração na especificação de fluxos e nas ações possíveis associadas a cada regra.

Nesta implementação foi utilizada uma característica dos comutadores HP Aruba, o qual permite usar a versão do protocolo OpenFlow 1.3 com apenas uma tabela de fluxos, configurando a tabela 0 como uma tabela única. Desta forma permite a utilização do OpenFlow 1.3 e todas as suas configurações e características com apenas uma tabela de encaminhamento.

A primeira observação feita, foi que após a configuração do OpenFlow nos comutadores, estes já não respondem a nenhum tráfego o qual não tenha uma regra explícita na tabela de encaminhamento. Isto ocorreu na tentativa de enviar um ping partir de um host para o comutador físico através das portas de tráfego. Antes da configuração do OpenFlow no comutador, este respondia ao ping normalmente, contudo após a configuração do protocolo OpenFlow esse comportamento já não existia. Por *default* ele faz o *drop* de pacotes cujo não exista nenhuma regra correspondente.

### 3.4 *intent* proativo

Configurou-se uma intenção de tráfego na rede com funcionamento proativo. Com o objetivo de se conseguir implementar um funcionamento proativo da rede em si, permitindo apenas o tráfego dos pacotes com regras de fluxo pré-instaladas. Os estudos feitos que conduziram a

implementação do modo proativo do *intent* foram os seguintes.

Foi configurado um *intent* máquina-a-máquina através do GUI do ONOS entre dois hosts físicos de IP 192.168.100.100 e 192.168.100.11. Na figura 3.3 podemos observar que após a configuração do *intent* o ONOS estabelece um caminho de comunicação entre os dois hosts, caminho tracejado amarelo, que passa pelos comutadores HP:of:00013821c7522cc0 e HP:of:0001104f58fa000. Também observa-se que o ONOS instala automaticamente as regras de fluxos contendo as especificações do tráfego, como o endereço IP de partida e chegada nas tabelas de encaminhamento dos comutadores que fazem parte do caminho entre os dois host.

Contudo mesmo tendo já sido instaladas todas as regras para o estabelecimento da comunicação entre os dois hosts, o ping entre eles ainda não funcionava. Somente após a ativação da app *reactive forwarding* o ping entre os dois hosts funcionou de ambos os lados.

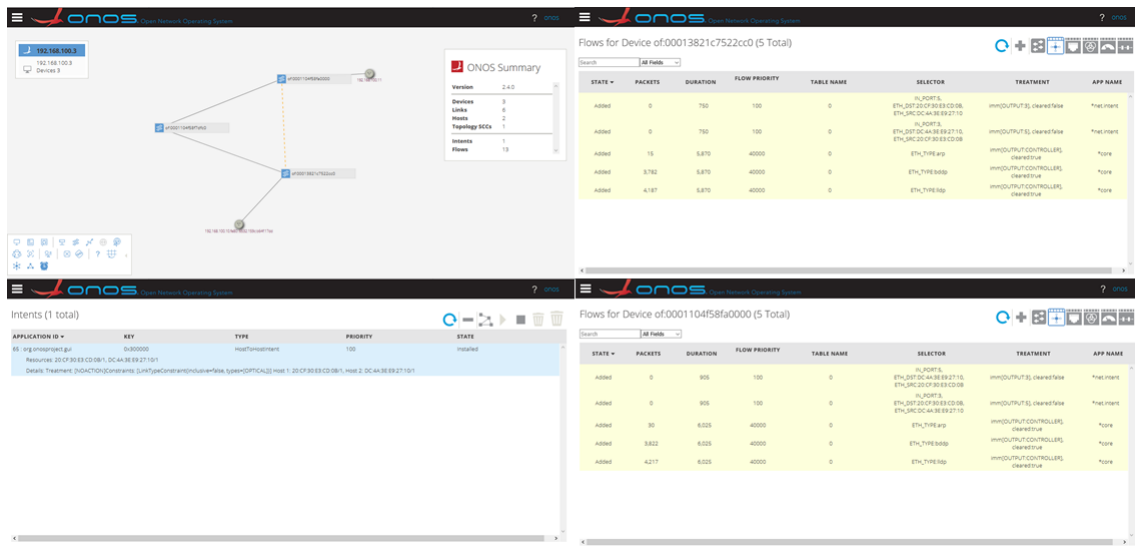


Figura 3.3: *intent* máquina-a-máquina

Este comportamento deve-se ao facto de o controlador ter instalado por defeito uma regra nos comutadores para os pacotes ARP, em que a ação associada a esta regra, fazia o envio do pacote para o controlador. No entanto, no controlador não estava ativo nenhuma App para fazer o tratamento dos pacotes ARPs, desta forma os dispositivos não se encontravam um ao outro. Por isso, após a ativação da App *reactive forwarding* que faz o encaminhamento de pacotes Internet Control Message Protocol (ICMP) e pacotes ARP de forma reativa conduziu ao funcionamento do *intent*.

A Application (App) *reactive forwarding* faz um encaminhamento reativo dos pacotes pela rede, o que não contribui para um funcionamento proativo da rede. Contudo, o ONOS possui também a app Proxy ARP/NDP. Esta aplicação responde a pedidos de ARP ou NDP em nome de um host de destino, se este host de destino for conhecido pelo controlador ONOS. O Proxy ARP considera a rede como uma rede de IP única, portanto se receber um pedido pelo qual não conhece o host de destino, inundará a rede com o pedido para que se o host de destino exista na rede e responderá.

Desta forma, fazendo a desativação da app *reactive forwarding* e a ativação do proxy ARP/NDP, possibilitou um funcionamento proativo do *intent* criado.

### 3.4.1 Corte de ligação direta

Na ocorrência de uma falha na ligação entre dois comutadores que fazem parte do caminho estabelecido pelo *intent*, o ONOS restabelece automaticamente a ligação entre os dois hosts, utilizando o próximo menor caminho entre eles. Sem que o administrador de redes faça qualquer configuração.



Figura 3.4: Falha na ligação entre comutadores

Podemos observar a figura 3.4, comparativamente a figura 3.3 anterior, que o controlador ONOS estabeleceu um novo trajeto entre os dois hosts. Pelo facto deste novo caminho passar pelo terceiro comutador, o ONOS faz a instalação automática das regras de fluxos neste novo comutador e as adaptações de portas necessárias nas regras nos comutadores anteriormente configurados.

### 3.4.2 Ligação dupla entre comutadores

Foi estabelecido uma ligação dupla entre os dois comutadores que fazem parte do caminho estabelecido pelo ONOS e este identifica as duas ligações como na figura 3.5.

Neste ambiente o *intent* continuou a funcionar pela ligação estabelecida inicialmente pela porta 3. No entanto ao fazer cair essa ligação, o *intent* automaticamente restabelece o seu tráfego de dados pela porta 7 nos mesmos comutadores.

## 3.5 Ligação rede virtual e física

A partir do desenvolvimento do ambiente de teste com equipamentos físicos, surgiu a ponderação quanto a escalabilidade do ambiente. Para isto foram feitas muitas análises e estudos dos *softwares*

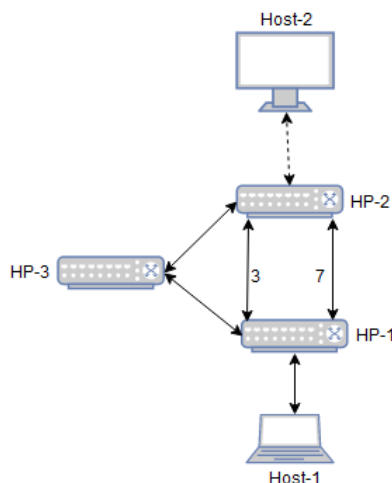


Figura 3.5: Falha na ligação dupla entre comutadores

e *hardwares* constituintes da rede. O primeiro passo foi a escolha de um emulador de rede que melhor se encaixa as necessidades de uma rede SDN.

Inicialmente, fez-se o uso do GNS3<sup>2</sup> para a emulação de uma rede constituída por comutadores Open vSwitches (OVS) e hosts virtuais. No entanto, no decorrer da implementação deparou-se com vários problemas relacionados, a máquina virtual, a configuração da conexão com o controlador e na conexão com a rede física.

Devido a esses problemas abandonou-se GNS3, investigou-se uma ferramenta amplamente utilizada para SDNs, o Mininet. Uma vez que este possui maior flexibilidade de configurações, os hosts podem correr comandos e programas pré-instalados e é melhor ter um host para as máquinas virtuais em ambiente Linux.

Deste modo fez-se a instalação do Mininet no ubuntu 20.04.2LTS através do terminal. Através de um script B.1 em Python, fez-se a criação de uma rede com 3 comutadores OVS e 5 hosts. Rede esta, que se encontra configurada e atribuída ao controlador ONOS através do mesmo script utilizado para o criar.

Contudo conforme pode-se observar na figura 3.6, através do controlador conseguiu-se observar a rede virtual e a rede física, mas estas não possuem uma ligação de comunicação entre elas.

Mesmo já tendo conseguido o aumento do número de dispositivos na rede, ainda não se obteve uma estrutura escalável e programática de modo a atender aos rigores dos ambientes SDN. Assim sendo, fez-se vários estudos, com o objetivo de se conseguir criar uma ligação entre a rede do Mininet constituída por comutadores OVS e a rede física constituída por comutadores HP Aruba.

<sup>2</sup>GNS3 - <https://gns3.com/>

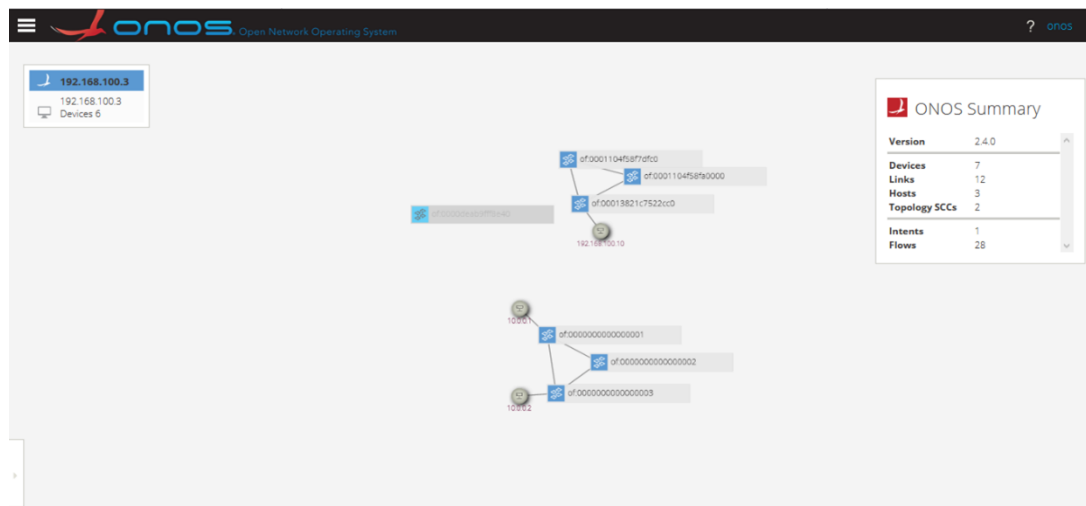


Figura 3.6: Rede física sem ligação com rede virtual

### 3.6 Configuração da ligação entre a rede física e a rede virtual

Para se conseguir a ligação e houver comunicação entre a rede virtual e a rede física utilizou-se um *script* B.1 de configuração, que é executado logo após a emulação da rede virtual. De salientar que de cada vez que a rede mininet for emulada, é preciso correr o *script* novamente de modo a se configurar a ligação entre as duas redes.

O *script* utilizado na implementação desta ligação, foi concebido após vários experimentos e testes descritos a seguir.

Primeiramente foram efectuados os seguintes procedimentos na tentativa de estabelecimento da ligação entre as duas redes. Configurou-se os IPs dos dispositivos das duas redes na mesma gama de endereços. Não tendo o resultado esperado, procedeu-se a configuração da rede Mininet de modo que os seus dispositivos emulados ficassem em *namespaces* diferentes, contudo o Open vSwitch não é suportado neste ambiente.

Posteriormente fez-se a configuração de uma das *bridges* do Mininet, a qual faz a ligação com o controlador e a rede física. Atribuindo um endereço IP e máscara a sua interface interna, contudo esta configuração levou a perda de conexão da rede virtual com o controlador. Tentou-se ainda a configuração de uma nova interface a mesma bridge, contudo ao tentar a conexão através desta nova interface, ela não era reconhecida.

A partir dos procedimentos apontados juntamente com as observações feitas concluiu-se que seria necessário a utilização da própria interface física da máquina, onde se encontrava o Mininet (interface `enp1s0`). Assim, adicionando-a a *bridge* e configurando uma *gateway* dessa *bridge* para a rede externa de modo a resolver o problema “*no route to host*”.

Contudo mesmo tendo todas essas configurações feitas não foram suficientes para ter a ligação entre as duas redes, de modo que o controlador conseguisse identificar e gerir as redes como uma

só.

Como nova tentativa, optou-se por configurar todas as comunicações dos comutadores OVS com o controlador para funcionamento in-band, uma vez que estava a ser utilizada apenas uma ligação para a comunicação com o controlador e para o tráfego de dados.

### 3.6.1 Experimento 1: Implementação

Rede virtualizada através do Mininet com comutadores Open vSwitches e rede física com comutadores HP Aruba.

#### Passos:

- Configurar uma *bridge* de um dos comutadores virtuais com IP e máscara e adicionar a interface `enp1s0` (interface física da máquina aonde se encontra o Mininet) a *bridge*.

```
echo "Configurando bridge switch s1 e Adicionando interface enp1s0 a bridge...."
sudo ovs-vsctl add-port s1 enp1s0
sudo ifconfig enp1s0 0
sudo ifconfig s1 192.168.100.11 netmask 255.255.255.0
sudo route add default gw 192.168.100.1 s1
sudo ip link set s1 up
```

Bloco de Código 3.1: Adicionar interface física dentro da *bridge* virtual

Comutador virtual: `s1`; *Bridge* utilizada: `s1`; Interface física: `enp1s0`;

- Configurar as *bridges* para funcionamento in-band (todas as bridges de cada comutador virtual).

```
echo "Configurando as bridges das switches para funcionamento in-band..."
sudo ovs-vsctl set bridge s1 other_config:disable-in-band=false
sudo ovs-vsctl set bridge s2 other_config:disable-in-band=false
sudo ovs-vsctl set bridge s3 other_config:disable-in-band=false
```

Bloco de Código 3.2: Configurar as *bridges* das switches para funcionamento in-band

- Configurar os comutadores para comunicar de forma *in-band* com o controlador (todos os comutadores virtuais).

A partir dos procedimentos adotados, observou-se a necessidade de criar um *script* com todas esses passos, uma vez que têm de ser feita cada vez que rede virtual é criada.

```
echo "Configurando connection-mode de cada switch para in-band.... "
CONTROLLER=$(sudo ovs-vsctl list controller | grep _uuid | awk '{print $3;}')
for arg in $CONTROLLER
do
    ovs-vsctl set controller $arg connection-mode=in-band
done
```

Bloco de Código 3.3: Configurar *connection-mode* de cada OVS para in-band

Com a implementação do *script* acima obteve se os seguintes resultados, o controlador já conseguia identificar as duas redes, a ligação entre elas e também já conseguia adicionar regras de fluxos a todos os dispositivos da rede. Pode se observar a topologia da rede na figura 3.7

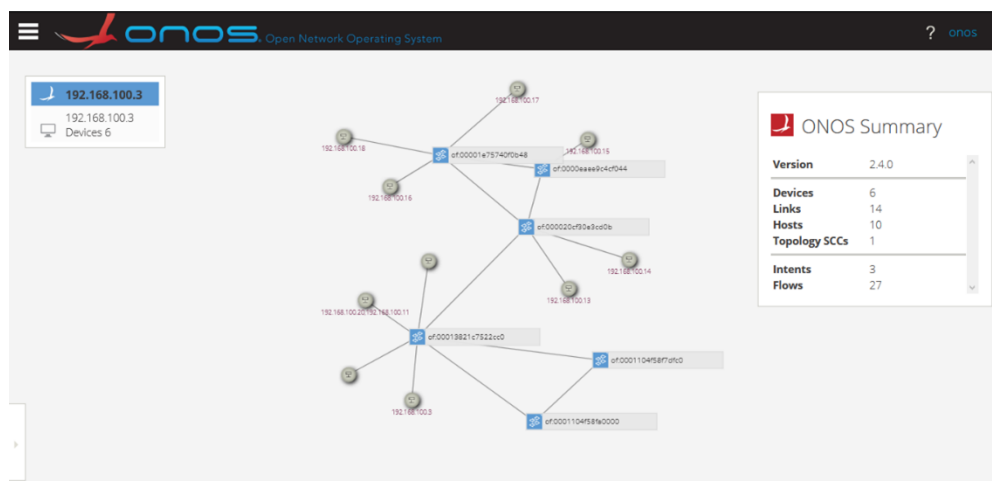


Figura 3.7: Topologia de rede incluindo a ligação entre a rede virtual e a rede física

Ao testar a comunicação na rede, verificou-se que o ping entre hosts da mesma rede (rede Mininet) funciona, contudo, o ping entre hosts da rede virtual e hosts da rede física não.

Com o auxílio do wireshark<sup>3</sup> observa-se que há trocas de pacotes arps entre os dois presentes na ligação, mas não há trocas de pacotes ICMP. Isto acontece, mesmo tendo a aplicação do *reactive forwarding* ativo ou criando um *intent* entre os hosts de modo a adicionar as regras de fluxos nos comutadores.

O único momento em que observou-se o funcionamento do ping é quando se encontra a app do *reactive forwarding* ativo e ao mesmo tempo um *intent* instalado entre os dois host. Esse resultado não condiz com o esperado, uma vez que era suposto esta comunicação funcionar só com um *intent* criado entre os hosts, sem a utilização da app do *reactive* de modo a ter um rede com funcionamento proativo.

<sup>3</sup>wireshark - <https://www.wireshark.org/>

### 3.6.2 Experimento 2: Estudo das apps

Devido a existência dos apps no controlador ONOS, que fazem o tratamento de alguns pacotes como o arp. Fez-se então alguns testes com estas apps na tentativa de identificar o problema do ping. Testes apresentados na tabela 3.1 correspondem a pings realizados entre o host virtual H1:192.168.100.13 e o host físico, respeitando a determinados critérios de testes.

O conjunto de testes realizados encontram-se sumarizados na tabela 3.1. Um dos aspectos a ter em conta é o facto de que mesmo tendo a app *reactive forwarding* ativada, sem a criação de *intent*, esta não encaminha pacotes icmp para hosts em redes diferentes (rede física e rede Mininet). Sabendo que no seu funcionamento normal, faz o transporte dos pacotes icmps estabelecendo deste modo a comunicação.

Tabela 3.1: Tabela de estudo sobre a influências de apps na ligação

|        | Atributos                                                                                                                                                   | Resultado                                                                                                                                                                                                                                                                                                                  |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Caso 1 | <ul style="list-style-type: none"> <li>• Com <i>intent</i>;</li> <li>• App proxy/arp ativo;</li> <li>• App <i>reactive forwarding</i> ativo;</li> </ul>     | <ul style="list-style-type: none"> <li>• O wireshark identifica – “duplicate use of 192.168.100.13 detected”(só aconteceu uma vez);</li> </ul>                                                                                                                                                                             |
| Caso 2 | <ul style="list-style-type: none"> <li>• Com <i>intent</i>;</li> <li>• App proxy/arp inativo;</li> <li>• App <i>reactive forwarding</i> inativo;</li> </ul> | <ul style="list-style-type: none"> <li>• Não há comunicação (ping) entre os host. O wireshark mostra que não há respostas a pacotes arps. O de esperar pois nem o <i>reactive forwarding</i> nem o proxy encontra-se ativo para tratar destes pacotes;</li> </ul>                                                          |
| Caso 3 | <ul style="list-style-type: none"> <li>• Com <i>intent</i>;</li> <li>• App proxy/arp ativo;</li> <li>• App <i>reactive forwarding</i> inativo;</li> </ul>   | <ul style="list-style-type: none"> <li>• Não funciona, há apenas trocas de pacotes arps. Desta forma era suposto funcionar pois a app proxy/arp trata dos pacotes arps, e as regras criadas pelos <i>intents</i> possibilita a passagem dos outros tipos de pacotes;</li> </ul>                                            |
| Caso 4 | <ul style="list-style-type: none"> <li>• Com <i>intent</i>;</li> <li>• App proxy/arp inativo;</li> <li>• App <i>reactive forwarding</i> ativo;</li> </ul>   | <ul style="list-style-type: none"> <li>• Já há troca de pacotes arps e pacotes icmps, há ping entre os hosts. Mesmo com a app proxy/arp desativada, há o tratamento dos pacotes arps, pois como observada anteriormente no decorrer deste projeto a app <i>reactive forwarding</i> também trata destes pacotes;</li> </ul> |

A regra do IPv4 nos comutadores físicos ficaram em *pending add*. Foram apagados e novamente instalados através da REST API da ONOS de modo a ficar no estado adicionado. A regra do IPv4 esta relacionada com a ativação da app *reactive forwarding*. Anteriormente, no decorrer deste projeto já se tinha observado que ao ativar a app *reactive forwarding* ele faz a instalação

da regra Internet Protocol Version 4 (IPv4) nos comutadores. No entanto, como desta vez os comutadores físicos estavam configurados com OpenFlow 1.3, estes ficaram no estado *pending add*. Contudo nos comutadores OVS foram configuradas correctamente, isto com a app *reactive forwarding* ativada. Desativando a app, ele apaga as regras IPv4 de todos os comutadores, incluindo as virtuais.

Após a configuração manual da regra do IPv4 nos comutadores físicos e virtuais, fez-se novamente os testes dos apps e obteve-se o mesmo resultado que os testes anteriores.

De notar também que mesmo tendo adicionado uma regra do IPv4 manualmente, após ativar o *reactive forwarding*, a app tenta adicioná-la novamente, e fica no estado *pending add* nos comutadores físicos. Já nos comutadores OVS ficam adicionadas as duas regras.

Outro aspecto importante a considerar, é que tendo o IPv4 completamente adicionado nos comutadores físicos, ou seja, no estado adicionado, já há comunicação entre as duas redes somente ativando a app do *reactive forwarding*, sem ter-se criado nenhum *intent* entre eles. Retirando a regra do IPv4 do comutador físico, o *reactive forwarding* já não funciona sem ter um *intent* criado.

Contudo, a comunicação por *intent* ainda não funciona sem o *reactive forwarding* mesmo com a regra do IPv4 instalada.

Em busca da resolução, fez-se mais testes de comunicação entre as duas redes observando as apps, a regra IPv4 e as tramas ethernet. Na ligação entre o host H1:192.168.100.13 e host físico de IP:192.168.100.10.

Os resultados obtidos encontram-se demonstrados na tabela 3.2. Incluindo os atributos particulares de cada ambiente de teste.

Apesar destes novos testes, terem apresentado aproximadamente os mesmos resultados que os anteriores, foram muito importantes porque permitiram identificar a causa deste problema de comunicação entre as duas redes.

Neste caso ao fazer ping do host virtual H1, com IP:192.168.100.13 ao host físico com IP:192.168.100.10, utilizando o wireshark no H1 observou-se que as tramas que chegam ao host virtual em resposta ao ARP request, contêm o protocolo 802.1Q, juntamente com o identificador da VLAN do host físico. A figura 3.8 ilustra a resposta.

No entanto, como observado anteriormente nos estudos dos VLANs, as Open vSwitches não estão configuradas em nenhuma VLAN (VLAN ID =0).

Procedeu-se então a observação dos pacotes ARPs através do wireshark no host físico. Este que também mantém-se no loop entre pedidos e respostas arps, no entanto, na observação das tramas constatou que estes não estavam etiquetadas.

Depois fez-se as mesmas observações no caso 4 na tabela 3.1, em que havia trocas de pacotes arps e pacotes icmp, e constatou-se que não havia nenhuma trama etiquetada, em nenhum dos

Tabela 3.2: Tabela de estudo sobre a influências da regra IPV4 na ligação

|        | Atributos                                                                                                                                                                         | Resultado                                                                                                                                                                                                                                                                                        |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Caso 1 | <ul style="list-style-type: none"> <li>Sem regra do IPV4;</li> <li>Sem <i>intent</i>;</li> <li>App proxy/arp inativo;</li> <li>App <i>reactive forwarding</i> ativo;</li> </ul>   | <ul style="list-style-type: none"> <li>O h1 faz o ping request mas este não chega no host físico. Só há trocas de pacotes arps;</li> </ul>                                                                                                                                                       |
| Caso 2 | <ul style="list-style-type: none"> <li>Sem regra do IPV4;</li> <li>Com <i>intent</i>;</li> <li>App proxy/arp inativo;</li> <li>App <i>reactive forwarding</i> inativo;</li> </ul> | <ul style="list-style-type: none"> <li>O h1 apresenta a mensagem <i>Destination host unreachable</i> e nem chega há enviar ping request. Pois não há resposta ao pedido arp, este não chega ao host físico (funcionamento normal, pois não há nenhuma app a tratar dos pacotes arps);</li> </ul> |
| Caso 3 | <ul style="list-style-type: none"> <li>Sem regra do IPV4;</li> <li>Com <i>intent</i>;</li> <li>App proxy/arp ativo;</li> <li>App <i>reactive forwarding</i> inativo;</li> </ul>   | <ul style="list-style-type: none"> <li>Há trocas de pedidos arps, mas ainda não há o envio do icmp request, aparece a mensagem <i>Destination host unreachable</i>;</li> </ul>                                                                                                                   |

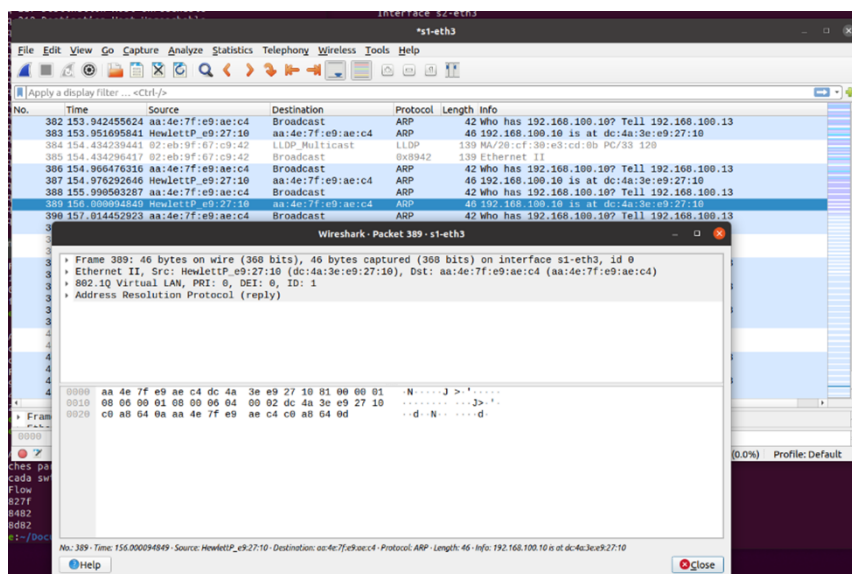


Figura 3.8: Print de resposta arp enviado pelo host físico, capturado no host virtual H1

lados.

### 3.6.3 Experimento 3: Configuração de VLANs

Devido a discrepância na marcação do VLAN ID dos pacotes de pedidos e respostas, procedeu-se ao estudo e configuração da mesma, com o objetivo de se ter todos dispositivos e hosts na mesma VLAN ID.

De acordo com a informação apresentada pelo controlador ONOS, não há nenhuma VLAN criada na rede do Mininet, enquanto que nos comutadores físicos encontram-se todas no VLAN 1. No entanto, na tentativa de resolver o problema tentou-se eliminar o “VLAN 1” existente nos comutadores físicos, de maneira em que toda a rede ficaria sem VLAN determinado. Contudo não foi possível a eliminação do VLAN dos comutadores físicos. Os comutadores HP Aruba retornavam sempre a mensagem “*VLAN 1 is the primary VLAN and cannot be deleted*”.

Passou-se então a configuração da VLAN nos comutadores virtuais do Mininet de modo em que todos ficassem na VLAN 1. No entanto sempre que se configurava a interface ligada a rede para ficar na VLAN 1, o controlador deixava de identificar a rede do Mininet. O mesmo acontecia na configuração das bridges para ficarem na VLAN 1, e por vezes o controlador apresentava erros implicando em reconfiguração.

Deste modo fez-se os seguintes testes com diferentes configurações e etiquetagem de tramas, com o objetivo de se ter uma forma de etiquetagem unânime em toda a rede.

#### 3.6.3.1 Descrição dos testes com VLANs

Inicialmente, o objetivo foi configurar as Open vSwitches de maneira que as tramas que chegassem ao host virtual não contessem as etiquetas 802.1Q, de modo a testar a conectividade da rede antes de avançar para a QoS, uma vez que as tramas etiquetadas constituem um papel importante para tal.

Deste modo, procedeu-se a configuração da ligação entre o host virtual H1:192.168.100.13 e o host físico de IP 192.168.100.10. Contudo, mencionado anteriormente, sempre que se configurava uma VLAN para a interface física `enp1s0`, que liga o comutador Open vSwitch ao comutador HP Aruba ou a *bridge* que a contém, o controlador deixava de identificar a rede.

Assim sendo, passou-se a configuração da interface `s1-eth3`, que é a interface ao qual se liga o host H1:192.168.100.13. Configurou-se a porta `s1-eth3` com `tag=1`, o mesmo VLAN ID configurada no comutador físico HP-2. Atribuiu-se o *VLAN mode = Access* de maneira que a interface `s1-eth3` funcionasse como uma porta de acesso, isto é, sempre que saísse uma trama retirara a sua tag e o host virtual iria receber a trama sem nenhuma tag. E ao entrar uma trama na interface o Open vSwitch etiquetava-o com a tag da porta.

No entanto ao testar a ligação, não se observou diferença alguma nas tramas, em nenhum dos hosts.

- **Tramas sem tag em novo VLAN ID**

Sendo o VLAN ID=1, o VLAN *default* dos comutadores HP Aruba, fez-se então a criação de um novo VLAN com ID = 100 nos comutadores físicos. Do mesmo modo fez-se a configuração dos comutadores OVS, interface s1-eth3 para tag=100 e VLAN *mode* = *Access*. Nos comutadores Aruba já não existe o VLAN *mode=access*, mas permitem ativar o IP routing, assim o comutador pode encaminhar o tráfego IPv4 [18].

Contudo, continuou-se a observar o mesmo comportamento nas ligações. Como se as configurações feitas nas Open vSwitches não estavam a fazer efeito algum. Mesmo fazendo outros testes com outros valores de *VLAN mode*, ou fazendo trunk, não se observou nenhuma diferença nas tramas em nenhum local da rede.

- **Configuração da interface física enp1s0**

Voltou-se então de novo o foco para a interface física que faz a ligação entre as duas redes. Apesar das tentativas falhadas de o configurar, procurou-se um pouco mais, a busca de uma solução para a configuração do seu VLAN. No entanto após algumas pesquisa encontrou-se uma solução.

Sabendo que a interface enp1s0 encontra-se configurada na *bridge* S1, e este não pode receber uma tag VLAN, a solução esta na configuração de uma *fake bridge* para a *bridge* S1 que contenha um identificador de VLAN.

Deste modo configurou-se uma *fake bridge* S10 com tag=100. Tendo a *fake bridge* criada, teve de configurar a interface física enp1s0 para esta nova *fake bridge*, deste modo a interface enp1s0 fica com tag=100, o mesmo da *fake bridge* em que se encontra.

Estas configurações levaram a algumas alterações no *script* de implementação da ligação. Também configurou-se o VLAN *mode* e atribui-se ao *other config* o valor de qinq-ethtype=802.1Q.

No entanto, após fazer todas essas alterações e configurações, as tramas que passam pela interface não apresentam nenhuma alteração esperada. Não coloca nem retira etiquetas das tramas.

Ao observar que as configurações de VLANs feitas até ao momento, não estavam a ter o resultado esperado, procurou-se saber o porquê. Foi então que se deparou com uma pergunta no FAQ do Open vSwitch para VLANs. A pergunta dizia, “*My OpenFlow controller doesn’t see the VLAN that I expect*” e na resposta encontrou-se, que as configurações de VLAN das Open vSwitches feitas através do ovs-vsctl, apenas afectam o tráfego implementados através do “*normal switching*”. Mas para os fluxos configurados através do controlador ou usando o of-ofctl, teríamos de configurar a manipulação de VLANs por meio das regras de fluxos.

Seguidamente, procurou-se outra forma de colocar tags na VLAN dos host virtuais. Desta forma utilizando uma *class* VLAN no *script* de emulação da rede. Fazendo com que as próprias

hosts enviassem tráfego já com as tags adicionadas. Além da nova classe no *script*, também foi necessário instalar o VLAN *package* na máquina ubuntu que contém a rede.

As tabelas 3.3 e 3.4 apresentam o resumo do conjunto de configurações e combinações realizadas.

Com as configurações efectuadas o controlador já identifica o VLAN ID, tanto dos host virtuais como os hosts físicos. Desta forma, as duas redes já conseguem trocar pacotes arps e identificar o endereço mac de cada host. Contudo após a troca de pacotes arps, nos quais todos continham a tag VLAN no header, o host virtual procedia com o envio de pacotes icmp *request* marcados com a tag VLAN, no entanto os icmp *reply* que retornavam do host físico chegavam sem tag. Assim permanecia neste loop de icmp *request* enviados pelo host virtual que não recebiam uma resposta.

#### 3.6.4 Experimento 4: Configuração OpenFlow Packet-in message

Após o conjuntos de experimentos realizados para tentar colocar as duas redes na mesma VLAN e ter uma ligação funcional, sem sucesso, voltou-se outra vez para explorar o modo sem VLAN para as duas redes. Uma vez que a rede virtual já se encontrava sem VLAN inicialmente, o problema aqui era retirar o VLAN ID dos hosts físicos. Após alguma observação mais a fundo, percebeu-se que o identificador de VLAN chegava ao controlador através das mensagens OpenFlow.

Deste modo procedeu-se a configuração dos comutadores HP Aruba, que estavam a enviar os *packet-in* com o VLAN ID. Alterou-se o valor do parâmetro OpenFlow *Packet-in VLAN tagging*, para *Untag Always*. Assim todas as mensagens que chegavam ao controlador não continham o VLAN ID. Configurações estas representadas na figura 3.9.

Desta forma, toda a rede ficou totalmente funcional. Já há comunicação entre a rede virtual e a rede física.

O objectivo principal de todos estes experimentos, é proporcionar escalabilidade ao cenário de teste. Deste modo veio a necessidade de incluir a rede mininet com as Open vSwitches, e configurá-los de maneira que resultassem numa só rede programável. Isto com o objetivo de colocá-lo a funcionar num modo proativo.

### 3.7 Topologia da rede híbrida

A figura 3.10 apresenta a topologia final da rede, onde as ligações a vermelho representam a ligação OpenFlow do canal seguro entre os comutadores e o controlador de rede, enquanto as ligações a preto representam os caminhos de tráfego de dados.

Na topologia apresentada podemos observar a diferença no modo de comunicação com o controlador pelas duas redes. Na rede física temos uma comunicação no modo out-of-band,

Tabela 3.3: Resumo dos testes com VLAN 1

| Testes                         | Atributos/Config:                                                                                                                                    | Resultado                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tramas sem tag                 | <ul style="list-style-type: none"> <li>Interface: S1-eth3;</li> <li>Tag = 1;</li> <li>VLAN mode = Access;</li> <li>HP Aruba VLAN = 1;</li> </ul>     | <ul style="list-style-type: none"> <li>Configurando esta interface como uma porta de acesso, todas as tramas deveriam chegar ao host final sem nenhum tag VLAN;</li> <li>No entanto os pacotes arps continuam a chegar ao host virtual com o VLAN ID do host físico;</li> <li>Não se observou efeito algum;</li> </ul>                                                                                                                   |
| Tramas sem tag em novo VLAN ID | <ul style="list-style-type: none"> <li>Interface: S1-eth3;</li> <li>Tag = 100;</li> <li>VLAN mode = Access;</li> <li>HP Aruba VLAN = 100;</li> </ul> | <ul style="list-style-type: none"> <li>Como VLAN ID = 1 é o <i>primary</i> VLAN do HP Aruba, alterou-se o VLAN ID tanto no comutador HP como na OVS;</li> <li>Não se observou efeito algum;</li> </ul>                                                                                                                                                                                                                                   |
| Interface física enp1s0        | <ul style="list-style-type: none"> <li>Interface: enp1s0;</li> <li><i>Fake bridge</i> = S10;</li> <li>Tag = 100;</li> </ul>                          | <ul style="list-style-type: none"> <li>Fez-se a criação de uma <i>fake bridge</i> e se configurou a interface enp1s0 dentro dela, de modo a ficar com o valor da tag = 100;</li> <li>Essas configurações levaram a alteração no script da rede;</li> <li>Testou-se diferentes VLAN <i>mode</i>;</li> <li>No entanto, apesar da interface física já pertencer a uma VLAN, não se observou nenhuma diferença na sua performance</li> </ul> |

Tabela 3.4: Resumo dos testes com VLAN 2

As configurações de VLAN das OpenvSwitches através do `ovs-vsctl` não tem efeito no controlador OpenFlow

|                                                |                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Script para colocar VLAN ID nos host virtuais. | <ul style="list-style-type: none"> <li>• Class VLAN;</li> <li>• VLAN <i>Package</i>;</li> </ul>               | <ul style="list-style-type: none"> <li>• Faz com que o próprio host virtual enviasse tráfego já com o tag adicionado na trama;</li> <li>• <b>Há trocas de pacotes arps;</b></li> <li>• O controlador já identifica o VLAN ID de todos os hosts;</li> <li>• <b>Problema:</b> Pedidos icmps enviados pelo host virtual vão com tag, e as resposta icmp vêm sem tag, deste modo fica num loop de pedidos sem respostas corretas ao pedido icmp;</li> </ul>                                  |
| <b>Retirar VLAN ID de toda a rede</b>          | <ul style="list-style-type: none"> <li>• Packet-in: <i>untag always</i> (HP Aruba Openflow config)</li> </ul> | <ul style="list-style-type: none"> <li>• A rede mininet já se encontrava sem VLAN id inicialmente;</li> <li>• Observou-se que as mensagens OpenFlow enviadas pelos comutadores HP Aruba continham tag do VLAN ID;</li> <li>• Configurou-se o packet-in para <i>untag always</i>;</li> <li>• As mensagens OpenFlow que chegam no controlador já não levam o VLAN ID;</li> <li>• A rede já se encontra totalmente funcional com a ligação entre a rede física e a rede virtual;</li> </ul> |

aonde cada comutador possui um caminho para a troca de mensagens com o controlador, fora do caminho de tráfego de dados.

Enquanto na rede virtual podemos observar que tanto a ligação entre o Open vSwitch S1 e o controlador e a ligação entre o S1 com o comutador Aruba HP-3, são executados através da mesma interface `enp1s0`, isto deve se ao facto que a máquina aonde se encontra emulada a rede virtual só possuir essa interface de rede. Deste modo a comunicação da Open vSwitch S1 e todos os outros comutadores virtuais, com o controlador ocorre no modo in-band. Pois ocorre na mesma ligação em que o tráfego de dados.

```

Aruba-2930M-24G-PoEP# show openflow instance aggregate

Configured OF Version      : 1.3
Negotiated OF Version      : 1.3
Instance Name              : aggregate
Data-path Description      : aggregate
Administrator Status       : Enabled
Member List                : VLAN 1, 100

Pipeline Model              : Standard Match
Listen Port                : None
Operational Status         : Up
Operational Status Reason  : NA
Datapath ID                : 00013821c7522cc0
Mode                       : Active
Flow Location               : Hardware and Software
No. of Hardware Flows      : 3

Hardware Rate Limit        : 0 kbps
Software Rate Limit        : 100 pps
Conn. Interrupt Mode       : Fail-Secure
Maximum Backoff Interval   : 60 seconds
Probe Interval             : 10 seconds
Hardware Table Miss Count  : NA
No. of Software Flow Tables : 1
Egress Only Ports          : None
Table Model                : Policy Engine and Software
Source MAC Group Table     : Disabled
Destination MAC Group Table : Disabled
Default Miss Action        : Drop
Packet-In VLAN Tagging    : Untag Always

Controller Id Connection Status Connection State Secure Role
-----
1 Connected Active No Master

```

Figura 3.9: Configuração OpenFlow packet-in HP Aruba Switch

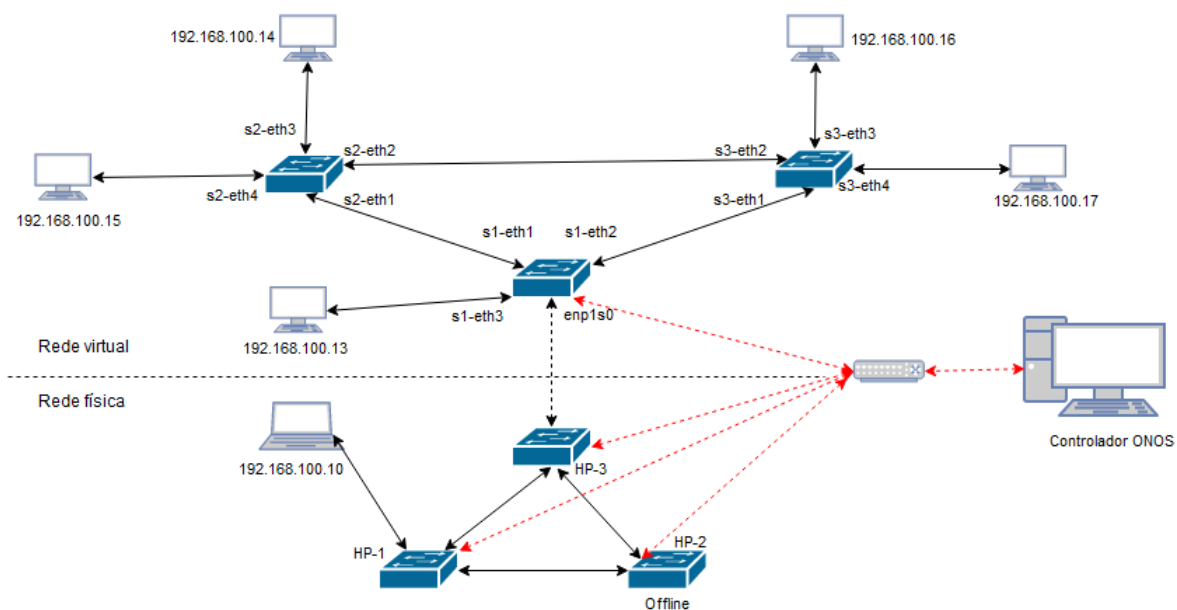


Figura 3.10: Topologia de rede híbrida

## Capítulo 4

# Implementação de QoS na rede

### 4.1 Configurações nos comutadores HP Aruba

Realizou-se a implementação do modelo de qualidade de serviço DiffServ nos comutadores físicos da HP Aruba, de modo a priorizar e atribuir maior largura de banda ao tráfego com marcação de maior prioridade. De modo a conseguir fazer esta implementação, fez-se as seguintes procedimentos.

A primeira configuração feita foi o ativação da transmissão do *Precision Time Protocol (PTP)* através das portas dos comutadores. Como podemos observar na figura 4.1, todas as portas já estão configuradas com o protocolo PTP, no modo *end-to-end transparent*.

Posteriormente procedeu-se a configuração das queues de saída em todas as portas dos comutadores, alterando a sua configuração default de *8-queues* para *4-queues*, sabendo que cada fila possui um valor de prioridade 802.1p apresentada na figura 4.2 e uma largura de banda minima reservada.

A largura de banda minima reservada para cada queue encontra-se apresentada na figura 4.3, no entanto no decorrer do desenvolvimento do projeto, este pode vir a ser alterado para 2-queues, uma vez que aloca maior largura de banda para cada queue.

Após as configurações das *queues*, o objetivo foi fazer o redirecionamento do tráfego para as respectivas *queues* de acordo com o seu valor de prioridade.

Deste modo a primeira abordagem foi criar *intents* com o identificador da *queue* para a qual o tráfego deve prosseguir. Pois o *intent* possui uma opção *-setQueue* que permite cria-lo desta forma, adicionando as regras de fluxos nos comutadores com o identificador da *queue* e da porta para o qual redirecionar o tráfego.

No entanto os fluxos com o identificador da *queue* ficaram em *pending-add* na tabela de fluxos dos comutadores HP. Isto deve-se ao facto do comutador HP Aruba 2930M não suportar esta configuração de OpenFlow [21].

```
Aruba-2930M-24G-PoEP(config)# show ptp
```

| Status and Counters - Precision Time Protocol |            |            |          |          |
|-----------------------------------------------|------------|------------|----------|----------|
| Port                                          | PTP Config | PTP Status | RX Count | TX Count |
| 1                                             | Enabled    | Active     | 0        | 0        |
| 2                                             | Enabled    | Active     | 0        | 0        |
| 3                                             | Enabled    | Active     | 0        | 0        |
| 4                                             | Enabled    | Active     | 0        | 0        |
| 5                                             | Enabled    | Active     | 0        | 0        |
| 6                                             | Enabled    | Active     | 0        | 0        |
| 7                                             | Enabled    | Active     | 0        | 0        |
| 8                                             | Enabled    | Active     | 0        | 0        |
| 9                                             | Enabled    | Active     | 0        | 0        |
| 10                                            | Enabled    | Active     | 0        | 0        |
| 11                                            | Enabled    | Active     | 0        | 0        |
| 12                                            | Enabled    | Active     | 0        | 0        |
| 13                                            | Enabled    | Active     | 0        | 0        |
| 14                                            | Enabled    | Active     | 0        | 0        |
| 15                                            | Enabled    | Active     | 0        | 0        |
| 16                                            | Enabled    | Active     | 0        | 0        |
| 17                                            | Enabled    | Active     | 0        | 0        |
| 18                                            | Enabled    | Active     | 0        | 0        |
| 19                                            | Enabled    | Active     | 0        | 0        |
| 20                                            | Enabled    | Active     | 0        | 0        |
| 21                                            | Enabled    | Active     | 0        | 0        |
| 22                                            | Enabled    | Active     | 0        | 0        |
| 23                                            | Enabled    | Active     | 0        | 0        |
| 24                                            | Enabled    | Active     | 0        | 0        |

Figura 4.1: Configuração PTP - HP Aruba Switch

```
Aruba-2930M-24G-PoEP(config)# show qos queue-config
```

| Egress Queue Configuration      |                   |
|---------------------------------|-------------------|
| Number of Queues : 4            |                   |
| Traffic Template : default-tcgt |                   |
| Queue                           | 802.1p Priorities |
| 1                               | 1-2               |
| 2                               | 0,3               |
| 3                               | 4-5               |
| 4                               | 6-7               |

```
Aruba-2930M-24G-PoEP(config)#
```

Figura 4.2: Configuração Queues - HP Aruba Switch

Os comutadores HP Aruba suportam dois tipos de QoS. O primeiro é o ip-precedence que faz a priorização dos pacotes IPv4 baseado nos valores ToS que estes carregam no cabeçalho. O segundo é o DiffServ que faz a priorização dos pacotes IPv4 baseado no valor DSCP que estes contêm.

De salientar que o HP Aruba já traz por *default* um valor de prioridade 802.1p ou valor CoS associado a cada DSCP codepoint como na figura 4.4. Deste modo tendo o DiffServ configurado e ativo nos comutadores HP Aruba, já apresentavam alguma qualidade de serviço em relação aos tráfego com prioridades diferentes. Testes feitos com o objetivo de se estudar o comportamento e

```
Aruba-2930M-24G-PoEP(config)# show bandwidth output
```

| Outbound Guaranteed Minimum Bandwidth Percentage |    |    |    |    |    |    |    |    |
|--------------------------------------------------|----|----|----|----|----|----|----|----|
| Port                                             | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
| 1                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 2                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 3                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 4                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 5                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 6                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 7                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 8                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 9                                                | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 10                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 11                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 12                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 13                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 14                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 15                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 16                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 17                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 18                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 19                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 20                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 21                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 22                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 23                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |
| 24                                               | 8  | 17 | 30 | 45 | *  | *  | *  | *  |

Figura 4.3: Largura de banda de cada queue - HP Aruba Switch

performance do DiffServ.

```
Aruba-2930M-24G-PoEP(config)# show qos type-of-service
```

DSCP-based Prioritization : Differentiated Services

| Codepoint | DSCP Policy | Priority |
|-----------|-------------|----------|
| 000000    |             | 0        |
| 000001    |             | 0        |
| 000010    |             | 0        |
| 000011    |             | 0        |
| 000100    |             | 0        |
| 000101    |             | 0        |
| 000110    |             | 0        |
| 000111    |             | 0        |
| 001000    |             | 1        |
| 001001    |             | 1        |
| 001010    |             | 1        |
| 001011    |             | 1        |

Figura 4.4: QoS Type of Service - HP Aruba Switch

A implementação da qualidade de serviço envolve três passos fundamentais, os quais são, selecionar o tráfego pretendido, marcá-lo com um valor DSCP e aplicar a política de qualidade pretendida. Contudo antes de marcar o tráfego selecionado com um determinado valor DSCP, este valor DSCP deve estar mapeado a uma prioridade 802.1p ou valor CoS.

Dentro de um comutador é o *Class of Service (CoS)* que decide qual a fila de prioridade deve

ser atribuída a um determinado pacote na sua saída. O valor DSCP no cabeçalho do pacote é transportado em cada salto, de um comutador para o outro para indicar a prioridade em que o pacote deve ser tratado, mas ao chegar num outro comutador este valor DSCP é mapeado num valor CoS correspondente, com base no qual é atribuída a fila de prioridade para este pacote.

A vantagem em transportar o nível de prioridade num valor DSCP é que este fica preservado em toda a rede. No caso, em que o transporte do nível de prioridade é feito no CoS, este é transportado no cabeçalho tag VLAN e deste modo pode ser perdido caso o pacote seja enviado para uma porta *untagged*.

Neste caso o primeiro passo é mapear os valores DSCP com os valores de prioridades locais pretendidos. Nos comutadores HP Aruba, estes já trazem por *default* os valores DSCP mapeados a prioridade correspondente, como apresentados na figura 4.5. Contudo estes podem ser alterados de acordo com os objetivos do administrador de rede.

```
Aruba-2930M-24G-PoEP(config)# show qos dscp-map
```

| DSCP Policies  |            |            |                  |
|----------------|------------|------------|------------------|
| DSCP CodePoint | DSCP Value | 802.1p tag | DSCP Policy name |
| 000000         | 0          | 0          | cs0              |
| 000001         | 1          | 0          |                  |
| 000010         | 2          | 0          |                  |
| 000011         | 3          | 0          |                  |
| 000100         | 4          | 0          |                  |
| 000101         | 5          | 0          |                  |
| 000110         | 6          | 0          |                  |
| 000111         | 7          | 0          | cs1              |
| 001000         | 8          | 1          |                  |
| 001001         | 9          | 1          |                  |
| 001010         | 10         | 1          | af11             |
| 001011         | 11         | 1          | af12             |
| 001100         | 12         | 1          |                  |
| 011111         | 31         | 3          | cs4              |
| 100000         | 32         | 4          |                  |
| 100001         | 33         | 4          |                  |
| 100010         | 34         | 4          | af41             |
| 100011         | 35         | 4          | af42             |
| 100100         | 36         | 4          |                  |
| 100101         | 37         | 4          | af43             |
| 100110         | 38         | 4          |                  |
| 100111         | 39         | 4          | cs5              |
| 101000         | 40         | 5          |                  |
| 101001         | 41         | 5          |                  |
| 101001         | 41         | 5          |                  |
| 101010         | 42         | 5          |                  |
| 101011         | 43         | 5          | ef               |
| 101100         | 44         | 5          |                  |
| 101101         | 45         | 5          |                  |
| 101110         | 46         | 7          |                  |
| 101111         | 47         | 5          |                  |

Figura 4.5: QoS dscp-map - HP Aruba Switch

O objetivo foi reservar a prioridade local máxima 7, apenas para o tráfego clock-PTP com marcação EF(46). Isto é, por *default* o comutador HP Aruba traz vários valores DSCP (56-63), mapeados com prioridade local 7. Contudo de modo a proporcionar ao tráfego PTP uma

prioridade EF(46) exclusiva de prioridade 7, fez-se o re-mapeamento dos valores DSCP (56-63) para a prioridade local 5.

O próximo objetivo é mapear a maior prioridade local 7, para a queue com maior largura de banda. No caso a fila com maior largura de banda é a 4.

O passo seguinte consiste em configurar o DSCP *trust mode* em todas as portas. A configuração *trust mode* tem como finalidade restringir quais informações da QoS devem ser utilizados para determinar o modo de entrada de um pacote no comutador e qual informação prioritária deve ser utilizada para prioridade local. A configuração da porta *trust mode* permite a preservação ou remoção das prioridades de QoS de entrada.

Os comutadores HP Aruba suportam os seguintes *trust mode*:

- **Device** - Em dispositivos aprovados, confiar no campo DSCP dos cabeçalhos em pacotes IP, e utilizar o dscp-map para remarcar a prioridade 802.1p. Se o codepoint DSCP não tiver uma prioridade associada, a prioridade será 0. Em dispositivos não aprovados, confiar na prioridade 802.1p e preservar quaisquer valores de IP-ToS;
- **DSCP** - Confiar no campo DSCP dos cabeçalhos dos pacotes IP, e utilizar o dscp-map para remarcar a prioridade 802.1p. Se o codepoint DSCP não tiver uma prioridade 802.1p associada, a prioridade será remarcada para 0.;
- **Ip-prec** - Confiar no campo IP-Precedence em pacotes IP e remarcar a prioridade 802.1p;
- **Dot1p** - Confiar na prioridade 802.1p e preservar o DSCP ou IP-ToS

Deste modo configurou-se o *trust mode* das 10 primeiras portas dos comutadores HP Aruba para DSCP, configurações estas representadas na figura 4.6.

```
Aruba-2930M-24G-PoEP(eth-2)# show qos trust
```

| Port-based QoS Trust Configuration |            |                    |
|------------------------------------|------------|--------------------|
| Port                               | Trust Mode | Device Trust State |
| 1                                  | Dscp       |                    |
| 2                                  | Dscp       |                    |
| 3                                  | Dscp       |                    |
| 4                                  | Dscp       |                    |
| 5                                  | Dscp       |                    |
| 6                                  | Dscp       |                    |
| 7                                  | Dscp       |                    |
| 8                                  | Dscp       |                    |
| 9                                  | Dscp       |                    |
| 10                                 | Dscp       |                    |
| 11                                 | Default    |                    |
| 12                                 | Default    |                    |
| 13                                 | Default    |                    |
| 14                                 | Default    |                    |

Figura 4.6: Port Trust Mode - HP Aruba Switch

Com estas configurações, os cumatores HP Aruba já fazem um tratamento diferenciado para tráfego com maior prioridade, enviando-os para a *queue* de maior prioridade e proporcionando-os maior largura de banda consequentemente.

## 4.2 Configurações nas Open vSwitches

No que diz respeito a qualidade de serviço na parte da rede virtual, o ideal seria a ativação do *differentiated service* nas Open vSwitches, contudo estas não suportam este mecanismo de qualidade de serviço.

Deste modo procedeu então a implementação manual da qualidade de serviço nos comutadores virtuais utilizando filas de menor e maior prioridade. O primeiro passo foi a configuração de duas filas de espera nas interfaces dos comutadores virtuais. Inicialmente utilizando o programa utilitário `ovs-vsctl` para configurar as Open vSwitches, configurou-se na interface `en1ps0`, que liga a rede virtual a rede física, duas filas, uma com o `min-rate=20000` bits e `max-rate=100000000` bits e o outro com o `min-rate=10000000` bits e `max-rate=100000000` bits.

A ideia inicial era criar uma regra de fluxo a partir de um *script*, e instalar esta regra apenas no Open vSwitch S1, o qual contém a interface `en1ps0`. De tal forma que a regra instalada redireciona-se o tráfego do *intent* que estabelece a conectividade, para a fila de maior largura de banda.

No entanto devido a conflitos nas especificações do *intent* máquina-a-máquina que cria o caminho para tráfego e a nova regra instalada, esta abordagem não teve o resultado esperado.

A solução encontrada para o redirecionamento do tráfego para a fila correta nas Open vSwitches foi a configuração `-setQueue` do *intent*. Com as filas criadas nas interfaces utilizadas e o *intent* ou regras de fluxos contendo o `setQueue` com o identificador da fila para qual redirecionar o tráfego, a qualidade de serviço desejada é obtida na rede virtual.

## 4.3 Qualidade de serviço na rede implementada

As configurações realizadas até o momento, para os comutadores físicos e para os comutadores virtuais, combinadas com um *intent* máquina-a-máquina já são suficientes para implementar a qualidade de serviço, separadamente, apenas para a rede virtual ou apenas para a rede física.

Para a rede física a configuração de um simples *intent* máquina-a-máquina para determinar o caminho de tráfego, já é aplicada a qualidade de serviço para tráfegos enviados com marcação DSCP de maior prioridade.

Para a rede virtual, a diferença está no *intent* criado para estabelecer o caminho de tráfego, pois este tem de ter a configuração `-setQueue` de modo a indicar para qual fila de prioridade o tráfego deve ser enviado, uma vez que as Open vSwitches não fazem o redirecionamento

automático baseado no DSCP.

De modo a ultrapassar esta limitação, e conseguir implementar a qualidade de serviço para toda a rede híbrida, fez-se o uso do *intent* ponto-a-ponto. Este tipo de *intent* faz a conexão de duas portas/pontos na rede de forma unidirecional.

Configurou-se *intents* diferentes para as duas partes da rede representada na figura 4.12. No lado da rede física criou-se um *intent* ponto-a-ponto entre a porta 10 do comutador HP-1, aonde se encontra conectado o host físico e a porta enp1s0 do Open vSwitch S1 que faz a fronteira para a rede virtual. De seguida criou-se um outro *intent* ponto-a-ponto entre as mesmas portas no sentido contrário, de maneira a existir uma comunicação bidirecional.

Do lado da rede virtual, também se configurou um *intent* ponto-a-ponto entre a porta enp1s0 do Open vSwitch S1 e porta 4(s3-eth4) da Open vSwitch S3, aonde se encontra conectado o host virtual pretendido. Repetiu-se as mesmas configurações no sentido contrário da ligação. Contudo estes *intents* na rede virtual contêm uma particularidade, eles têm a configuração *setQueue* que especifica por qual fila de saída deve ser redirecionado o tráfego.

Com estas implementações e utilização do *intent* ponto-a-ponto obtém-se uma qualidade de serviço global em toda a rede híbrida.

## 4.4 Injeção de tráfego

Nos testes de funcionamento do *intent*, foi utilizada a ferramenta Iperf<sup>1</sup> para a injeção de tráfego na rede. Funciona nas duas pontas da ligação, numa ponta como cliente e na outra como servidor, também fornece dados estatísticos do tráfego simulado.

- **Host para controlador**

Neste teste foi feito o envio de tráfego pelo iperf e ping, a partir de um host físico ligado a um dos comutadores, para o controlador. Contudo estas tentativas de comunicação falharam, mesmo com a criação de uma regra de fluxo para a ligação entre os dois. Isto deve ao facto de o controlador estar ligado ao comutador através da porta OOBM, deste modo não há ligação entre o controlador e as portas de tráfego do comutador aonde esta conectado o host.

- ***intent* Máquina-a-Máquina**

Neste teste foi utilizado o *intent* anteriormente configurado entre os hosts físico ubuntu 16.04 de IP 192.168.100.11 e o Windows 10 de IP 192.168.100.10. Realizou a primeira injeção de tráfego na rede, utilizando o Iperf, desta forma podendo observar as estatísticas do tráfego enviado. Com este teste conclui-se que a rede já se encontra completamente funcional.

---

<sup>1</sup>Iperf - <https://iperf.fr/>

- ***intent* Ponto-a-Ponto**

Foi configurado um *intent* ponto-a-ponto entre duas portas de entrada de dois comutadores diferentes. No entanto o tráfego iperf e ping entre os hosts ligados a estas portas não funcionavam. Porque o *intent* ponto-a-ponto funciona apenas numa direção. Daí a necessidade de configurar um outro *intent* entre as mesmas portas no sentido contrário.

Após a configuração do novo *intent* para a comunicação no sentido contrário, o tráfego entre os dois hosts funcionou normalmente.

No tipo de *intent* ponto-a-ponto também há a recompilação automática do *intent* após a ocorrência de um evento na rede (perda de link). Descobrindo assim um novo caminho para estabelecer a conexão, alterando as regras de fluxos de acordo com o novo caminho.

## 4.5 QoS: Intent + setDSCP

Este teste foi realizado com objetivo de estudar a funcionalidade e a performance setDSCP do *intent* na rede híbrida, uma vez que tanto os Open vSwitches como os comutadores HP Aruba suportam a configuração setDSCP.

Foi criado um *intent* máquina-a-máquina entre o host virtual H5:192.168.100.17 e o host físico de IP 192.168.100.10. Na figura 4.7 podemos observar o caminho definido pelo ONOS entre os dois hosts.

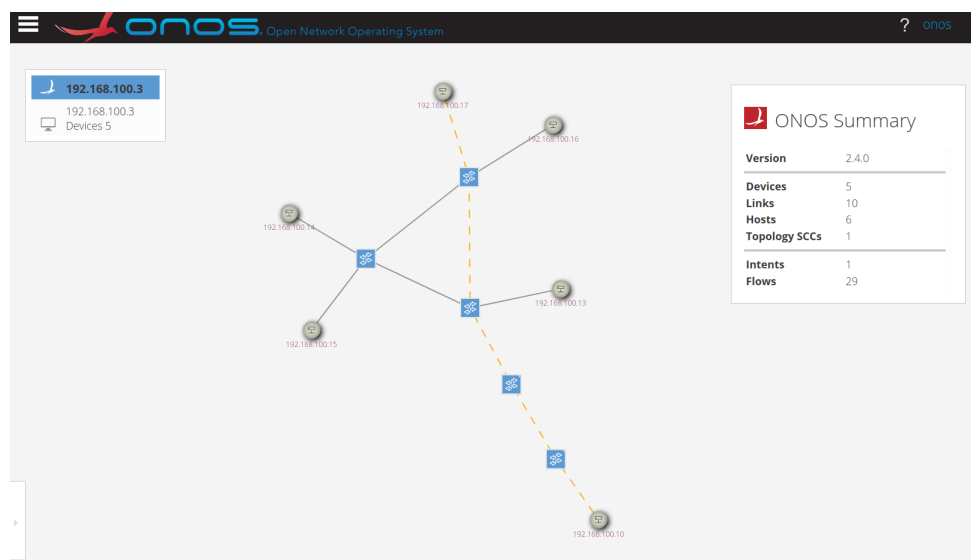


Figura 4.7: Caminho *intent* + DSCP

As especificações utilizadas no *intent* foram as seguintes:

Critérios:

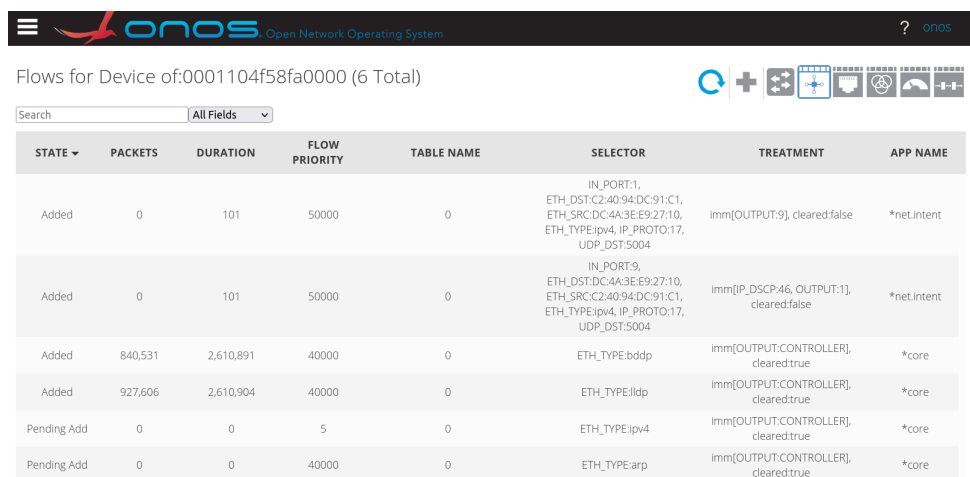
- Priority: 5000
- MAC Source: DC:4A:3E:E9:27:10
- MAC Destination: DE:EC:27:9B:93:E2
- ETH\_TYPE: IPv4
- IP\_PROTOCOL: 17 (UDP)

Tratamento:

- Type: L3MODIFICATION
- Subtype: IP\_DSCP
- ipDSCP: 46 (EF)
- ETH\_TYPE: IPv4
- IP\_PROTOCOL: 17 (UDP)

A abordagem aqui utilizada, foi criar um *intent* entre os dois hosts, no qual todo o tráfego UDP que chegasse no comutador, seria atribuído um valor DSCP 46 (EF), proporcionando maior prioridade. Isto aconteceria em todos os saltos de comutador para comutador, ou seja, logo ao chegar no primeiro comutador, este atribuída o valor DSCP. Chegando ao próximo comutador já com a prioridade atribuída e mantendo-a por todo o caminho definido pelo *intent*.

No entanto ao verificar as regras de fluxos instaladas nos comutadores, percebeu-se que este *intent* apenas altera o IP\_DSCP do tráfego, no último comutador, antes de chegar ao host de destino. Fazendo com que o tráfego chegasse ao destino com o DSCP 46, no entanto, durante todo o caminho este tem a sua prioridade inicial .



| STATE       | PACKETS | DURATION  | FLOW PRIORITY | TABLE NAME | SELECTOR                                                                                                              | TREATMENT                                   | APP NAME    |
|-------------|---------|-----------|---------------|------------|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------|-------------|
| Added       | 0       | 101       | 50000         | 0          | IN_PORT:1,<br>ETH_DST:C2:40:94:DC:91:C1,<br>ETH_SRC:DC:4A:3E:E9:27:10,<br>ETH_TYPE:ipv4, IP_PROTO:17,<br>UDP_DST:5004 | imm[OUTPUT:9], cleared:false                | *net.intent |
| Added       | 0       | 101       | 50000         | 0          | IN_PORT:9,<br>ETH_DST:DC:4A:3E:E9:27:10,<br>ETH_SRC:C2:40:94:DC:91:C1,<br>ETH_TYPE:ipv4, IP_PROTO:17,<br>UDP_DST:5004 | imm[IP_DSCP:46, OUTPUT:1],<br>cleared:false | *net.intent |
| Added       | 840,531 | 2,610,891 | 40000         | 0          | ETH_TYPE:bddp                                                                                                         | imm[OUTPUT:CONTROLLER],<br>cleared:true     | *core       |
| Added       | 927,606 | 2,610,904 | 40000         | 0          | ETH_TYPE:lldp                                                                                                         | imm[OUTPUT:CONTROLLER],<br>cleared:true     | *core       |
| Pending Add | 0       | 0         | 5             | 0          | ETH_TYPE:ipv4                                                                                                         | imm[OUTPUT:CONTROLLER],<br>cleared:true     | *core       |
| Pending Add | 0       | 0         | 40000         | 0          | ETH_TYPE:arp                                                                                                          | imm[OUTPUT:CONTROLLER],<br>cleared:true     | *core       |

Figura 4.8: Tabela de fluxos comutador HP

Na figura 4.8 temos a tabela de fluxos do comutador HP Aruba no qual está ligado o host físico na porta 4.

Podemos observar na especificação do tratamento que apenas o tráfego que sai na porta 4, ou seja, chega no host físico é marcado com o `IP_DSCP : 46`, no entanto o tráfego que entra na rede, não sofre nenhuma alteração.

Na figura 4.9, temos as regras observada apartir do CLI do comutador HP Aruba. No qual podemos observar no campo *apply actions*, *incoming port*, *output* as especificações efectuadas.

|                         |                    |                  |                 |                         |                    |                  |                 |
|-------------------------|--------------------|------------------|-----------------|-------------------------|--------------------|------------------|-----------------|
| Flow 5                  |                    |                  |                 | Flow 4                  |                    |                  |                 |
| Match                   |                    |                  |                 | Match                   |                    |                  |                 |
| Incoming Port           | : 2                | Ethernet Type    | : IP            | Incoming Port           | : 4                | Ethernet Type    | : IP            |
| Source MAC              | : deec27-9b93e2    | Destination MAC  | : d04a3e-e92710 | Source MAC              | : d04a3e-e92710    | Destination MAC  | : deec27-9b93e2 |
| Source MAC Mask         | : ffffffff         |                  |                 | Source MAC Mask         | : ffffffff         |                  |                 |
| Destination MAC Mask    | : ffffffff         |                  |                 | Destination MAC Mask    | : ffffffff         |                  |                 |
| VLAN ID                 | : Any              | VLAN Priority    | : Any           | VLAN ID                 | : Any              | VLAN Priority    | : Any           |
| Source IP Address       | : Any              |                  |                 | Source IP Address       | : Any              |                  |                 |
| Destination IP Address  | : Any              |                  |                 | Destination IP Address  | : Any              |                  |                 |
| IP Protocol             | : UDP              | IP DSCP          | : Any           | IP Protocol             | : UDP              | IP DSCP          | : Any           |
| IP ECN                  | : Any              | Destination Port | : Any           | IP ECN                  | : Any              | Destination Port | : Any           |
| Source Port             | : Any              |                  |                 | Source Port             | : Any              |                  |                 |
| Source Port Range       | : NA               |                  |                 | Source Port Range       | : NA               |                  |                 |
| Destination Port Range  | : NA               |                  |                 | Destination Port Range  | : NA               |                  |                 |
| TCP Flags               | : NA               |                  |                 | TCP Flags               | : NA               |                  |                 |
| TCP Mask                | : NA               |                  |                 | TCP Mask                | : NA               |                  |                 |
| Custom Match One        | : Any              |                  |                 | Custom Match One        | : Any              |                  |                 |
| Custom Match One Mask   | : Any              |                  |                 | Custom Match One Mask   | : Any              |                  |                 |
| Custom Match Two        | : Any              |                  |                 | Custom Match Two        | : Any              |                  |                 |
| Custom Match Two Mask   | : Any              |                  |                 | Custom Match Two Mask   | : Any              |                  |                 |
| Custom Match Three      | : Any              |                  |                 | Custom Match Three      | : Any              |                  |                 |
| Custom Match Three Mask | : Any              |                  |                 | Custom Match Three Mask | : Any              |                  |                 |
| Custom Match Four       | : Any              |                  |                 | Custom Match Four       | : Any              |                  |                 |
| Custom Match Four Mask  | : Any              |                  |                 | Custom Match Four Mask  | : Any              |                  |                 |
| Attributes              |                    |                  |                 | Attributes              |                    |                  |                 |
| Priority                | : 50000            | Duration         | : 841 seconds   | Priority                | : 50000            | Duration         | : 841 seconds   |
| Hard Timeout            | : 0 seconds        | Idle Timeout     | : 0 seconds     | Hard Timeout            | : 0 seconds        | Idle Timeout     | : 0 seconds     |
| Byte Count              | : NA               | Packet Count     | : 85038         | Byte Count              | : NA               | Packet Count     | : 2             |
| Flow Table ID           | : 0                | Controller ID    | : 1             | Flow Table ID           | : 0                | Controller ID    | : 1             |
| Cookie                  | : 0xbe0000cfa77e48 |                  |                 | Cookie                  | : 0xbe0000a3dd9a9d |                  |                 |
| Hardware Index          | : 0                |                  |                 | Hardware Index          | : 0                |                  |                 |
| Instructions            |                    |                  |                 | Instructions            |                    |                  |                 |
| Apply Actions           |                    |                  |                 | Apply Actions           |                    |                  |                 |
| Modify IP DSCP          | : 46               |                  |                 |                         |                    |                  |                 |
| Output                  | : 4                |                  |                 | Output                  | : 2                |                  |                 |

Figura 4.9: HP-1 - ação setDSCP

## 4.6 QoS na rede virtual: Intent + setQueue

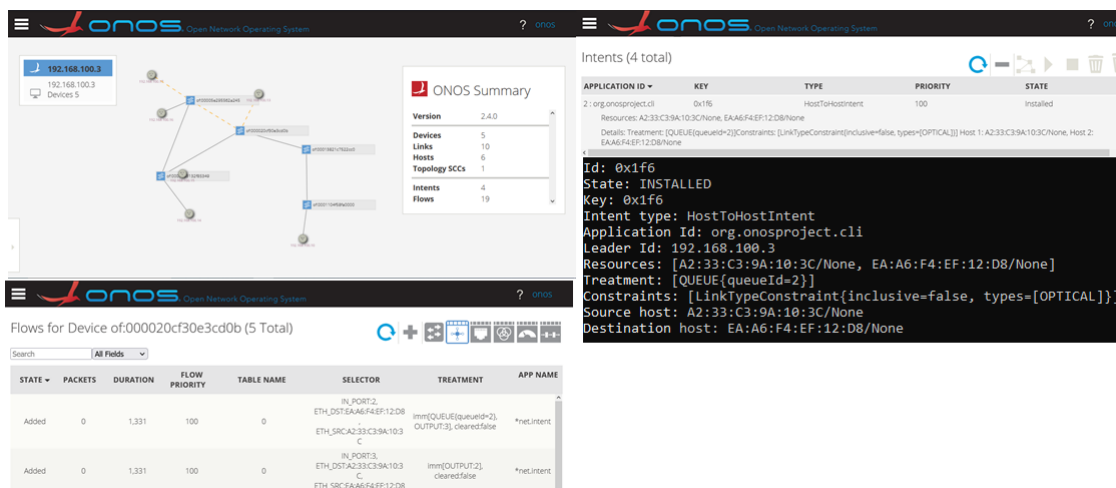
Este teste foi efetuado com o objetivo de provar a funcionalidade da qualidade serviço na rede virtual, utilizando o *intent* e a sua funcionalidade *setQueue*. Já que os comutadores Aruba não suportam a configuração *setQueue*.

De modo a implementar a QoS entre o host virtual h1:192.168.100.13 e o outro host virtual h5:192.168.100.17, primeiramente, foram configuradas as filas de prioridades nas Open vSwitches ao qual estão conectados os respetivos hosts utilizando o *script queueConfig* apresentado no apêndice.

De seguida foi criado um *intent* máquina-a-máquina utilizando o ONOS CLI com o comando: `Add-Host-intent -setQueue 2 A2:33:C3:9A:10:3C/None EA:A6:F4:EF:12:D8/None`. Em que o comando especifica a *queue* 2, como a *queue* para onde o tráfego deve ser encaminhado.

Na figura 4.10 podemos observar o *intent* criado através do GUI e do CLI do ONOS, também o caminho estabelecido por este, e as regras instaladas no Open vSwitch S1.

Na figura 4.11 podemos ver do lado esquerdo a estatísticas das filas das portas antes do envio de tráfego, e do lado direito são as estatísticas após a troca de tráfego entre os dois hosts.

Figura 4.10: Configuração *intent* + *setQueue*

Podemos observar que o tráfego está a ser encaminhado para a fila pretendida a *queue 2*.

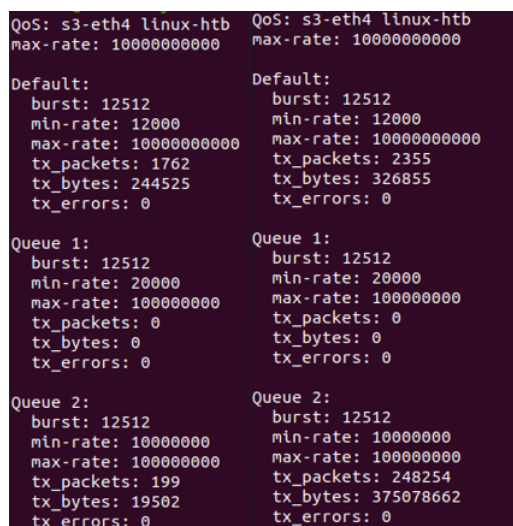


Figura 4.11: Estatísticas Queues porta s3-eth4

Com estas observações prova-se o funcionamento da qualidade de serviço implementada utilizando o *intent* e o *setQueue*.

Contudo uma outra observação no funcionamento do *setQueue* do *intent*, é que este tem um comportamento semelhante ao *setDSCP*. Não especifica em cada comutador que faz parte do caminho a *queue* para o qual o tráfego deve ser enviado, mas somente nos comutadores na ponta, aonde estão conectados os hosts.



físico 192.168.100.10. Este tráfego enviado durante 13s tem a largura de banda de 10Mbit/s e não traz marcação DSCP. Passa pela *queue* de menor prioridade nos comutadores virtuais.

Durante o envio do primeiro tráfego foi enviado outro tráfego a partir do host virtual h5:192.168.100.17 para o mesmo host físico. Esse envio foi feito durante 4s, para a *queue* de maior prioridade na parte da rede virtual, com a largura de banda de 10Mbit/s e com marcação DSCP de 46.

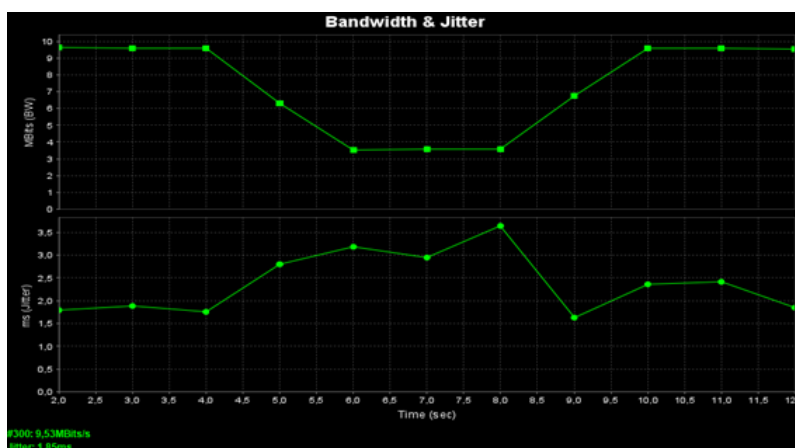


Figura 4.13: Largura de Banda e Jitter de tráfego sem prioridade 1.0

A figura 4.13 apresenta os efeitos da QoS implementada na rede. Ele representa o primeiro tráfego enviado sem prioridade. Pode-se observar que apartir do 4 segundo na duração da propagação do tráfego sem prioridade, a largura de banda providenciada para este tráfego começa a diminuir. Isto deve-se a entrada do outro tráfego com maior prioridade na rede. Esta diminuição de largura de banda nota-se de forma mais acentuada entre o 5 segundo e 9 segundo, o qual representa o período de 4 segundos de duração do tráfego de maior prioridade. Durante este mesmo período pode-se analisar o aumento considerável do Tremor (*jitter*), pois representa o periodo de maior conflito entre os pacotes dos dois tráfegos na competição de largura de banda.

Contudo estes dados provam que a maior largura de banda de banda foi alocada ao tráfego de maior prioridade. Sendo que o tráfego de menor prioridade, analisado no gráfico, sofre grandes perdas de pacotes, devido a pouca largura de banda disponível na porta ser atribuída na sua maioria para o tráfego de maior prioridade.

Este teste foi realizado algumas vezes e teve-se como resultados dois comportamentos parecidos. Há momentos em que o tráfego de maior prioridade recebe a largura de banda total da porta do comutador HP com 10Mbit/s, e há vezes em que este não recebe a percentagem total da largura de banda da porta, como no caso da figura 4.13 em que recebeu cerca de 65%. Sabendo que os dois tráfegos têm largura de banda 10Mbit/s, o suficiente para encher a largura disponível. Contudo este comportamento pode estar relacionado com a forma que os comutadores Aruba tratam o tráfego que passam por queues diferentes, sendo assim um ponto de estudo futuro.

- **Teste 2.0: Largura de banda semelhante a rede virtual**

Neste teste a porta no comutador HP-1 que recebe os tráfegos vindos da rede, foi aumentada para 100Mbit/s de modo a ter mesma largura de banda que a rede virtual.

Foi enviado um tráfego a partir do host virtual h3 para o host físico 192.168.100.10, representado na figura 4.16 pela cor roxo. Este tráfego enviado durante 13s tem a largura de banda de 100Mbit/s e não traz marcação DSCP. Passa pela *queue* de menor prioridade nos comutadores virtuais.

Durante o envio do primeiro tráfego foi enviado outro tráfego a partir do host virtual h5 para o mesmo host físico, representado na figura 4.16 na pela cor verde. Esse envio foi feito durante 4s, com a largura de banda de 80Mbit/s e com marcação DSCP de 46.

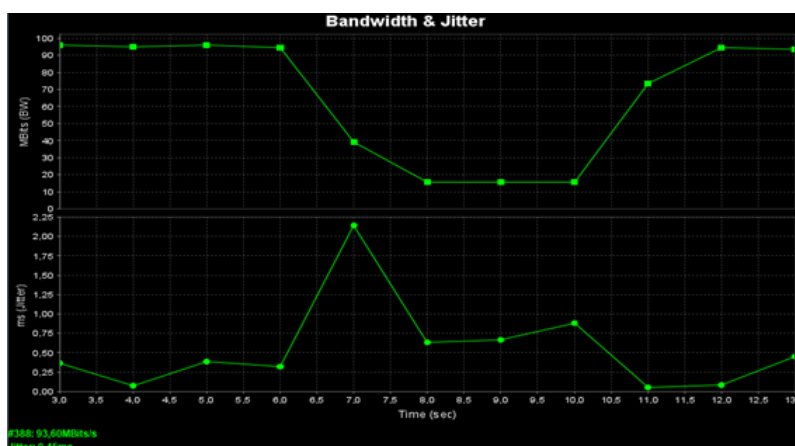


Figura 4.14: Largura de Banda e Jitter de tráfego sem prioridade 2.0

A figura 4.14 representa o primeiro tráfego enviado sem prioridade com largura de banda de 100Mbit/s, contudo no decorrer do tráfego podemos observar que a sua largura de banda foi reduzida cerca de 80%, ficando aproximadamente nos 20Mbit/s e o jitter teve um aumento brusco. O período em que isto acontece representa o período em que o segundo tráfego de maior prioridade foi enviado. Desta forma a maior largura de banda foi alocada para o tráfego de maior prioridade.

Neste caso podemos observar que os 80% de largura de banda retirada ao tráfego de menor prioridade, foi atribuída ao tráfego de maior prioridade, uma vez que este tem 80Mbit/s de largura de banda e o caminho de tráfego tem um total de 100Mbit/s de largura de banda. Desta forma conclui-se que a largura de banda está a ser providenciada de acordo com a necessidade do tráfego de maior prioridade.

### • Teste 3.0: Tráfego com maior prioridade

Neste teste a porta no comutador HP-1 que recebe os tráfegos vindos da rede, está limitada a 100Mbit/s. Realizado de forma contrária em relação aos testes anteriores.

Foi enviado um tráfego a partir do host virtual H5 para o host físico 192.168.100.10, representado na figura 4.16 pela cor verde, durante 13s, com largura de banda de 80Mbit/s, com

marcação DSCP 46 e com passagem pela *queue* de maior prioridade na rede virtual. Durante o envio do primeiro tráfego foi enviado outro tráfego, representado pela cor roxo, a partir do host virtual h3 para o mesmo host físico durante 4s e com largura de banda igual a 100Mbit/s e sem marcação DSCP.

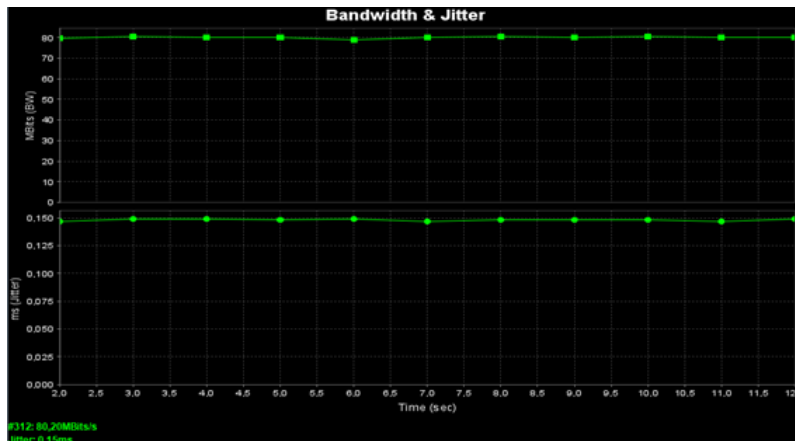


Figura 4.15: Largura de Banda e Jitter de tráfego com prioridade 3.0

Na figura 4.15 podemos observar o comportamento do primeiro tráfego enviado, o com maior prioridade. Nota-se que tem um largura de banda de 80Mbit/s, durante todo o seu período de propagação. E um Tremor muito baixo.

Mesmo havendo um outro tráfego em simultâneo na rede a competir durante 4 segundos, o tráfego de maior prioridade não sofre qualquer efeito em relação a sua largura de banda e tremor, que permanecem constante. E também não sofre nenhuma perda de pacote causado pelo tráfego de menor de prioridade.

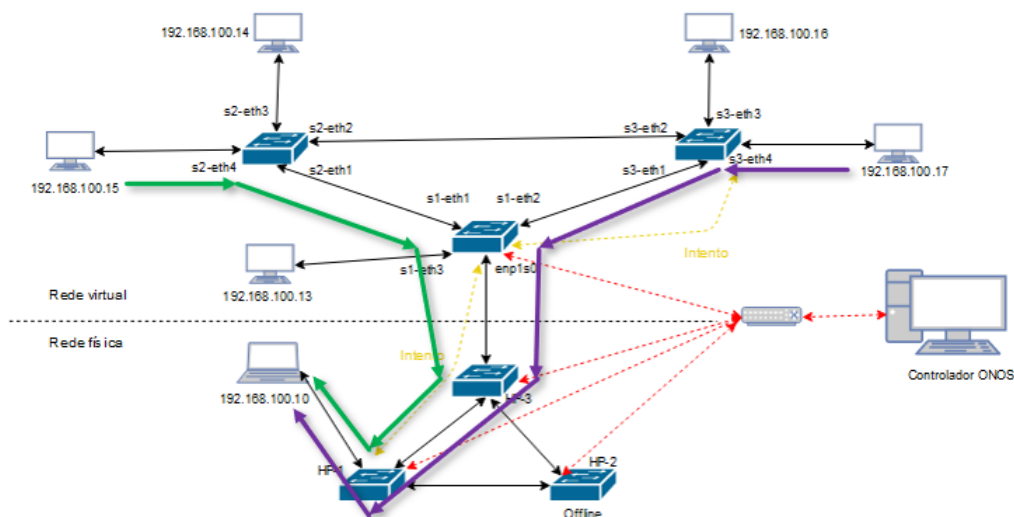


Figura 4.16: Propagação de tráfego



## Capítulo 5

# Conclusões

A rede tradicional vem enfrentando muitos desafios na medida que as tecnologias evoluem, a rede SDN traz a solução para alguns destes problemas. Contudo ainda há desafios a serem ultrapassados e melhorados, principalmente no que diz respeito a qualidade de serviço.

No decorrer da realização deste projeto, constatou-se que a rede SDN possui vários componentes que suportam a qualidade de serviço, contudo cada um ao seu nível e ambiente. Apesar do protocolo OpenFlow, ainda se constata algumas limitações na configuração e gestão dos dispositivos na interface *southbound*, principalmente quando estes são de fornecedores diferentes.

A abordagem DiffServ por exemplo, seria facilmente implementada numa rede com o controlador ONOS e comutadores HP Aruba. O mais trabalhoso nesta abordagem seria a ativação manual do *Differentiated Service* nos dispositivos HP. Tendo isto, do resto tudo pode ser configurado no controlador, utilizando o protocolo OpenFlow juntamente com o *intent* framework. A configuração desta mesma abordagem num ambiente com o Open vSwitch seria mais trabalhoso, uma vez que tinha de se efetuar mais configurações manuais nas Open vSwitches.

No entanto devido a sua programabilidade e flexibilidade, a rede SDN permite aos administradores de rede a oportunidade de criar e implementar uma qualidade de serviço diferente das abordagens existentes atualmente, proporcionando uma capacidade de inovação muito grande, como é o caso das aplicações no controlador ONOS.

No decorrer da realização deste projeto deparou-se com várias abordagens e perspectivas interessantes da rede SDN utilizando o protocolo OpenFlow. Em particular, no ambiente híbrido com dispositivos físicos como o HP Aruba e dispositivos virtuais como o Open vSwitch. A rede SDN mostra ser uma excelente abordagem para a solução de vários problemas na rede, particularmente para o tráfego de multimídia, que exige cada vez mais, uma melhor implementação da qualidade de serviço.

A rede definida por *software* vai de encontro com as limitações do protocolo AES67, no que diz respeito ao gerenciamento, configuração da rede e descoberta de novos dispositivos conectados.

O subsistema do *intent* disponibilizado pelo ONOS, consiste numa enorme vantagem na configuração do tráfego na rede SDN com qualidade de serviço principalmente pelas suas características particulares como:

- Permite estabelecer um caminho de tráfego quase de forma instantânea em toda a rede.
- Seu funcionamento em modo recursivo e com suporte a falhas nas ligações e desconexão de dispositivos.
- Os seus campos de critérios que permitem especificar um único tráfego.
- Além de estabelecer a conexão de tráfego, também harmoniza as políticas de qualidade de serviço por toda a rede, como o *setQueue* e *setDSCP* que foram experimentados neste projeto.
- Além destes dois ele suporta outras configurações de QoS como, a latência e a largura de banda para o tráfego implementado, contudo estes não tiveram nenhum efeito nos testes realizados, ou por falta de compatibilidade com os comutadores utilizados, ou por ser uma opção em desenvolvimento.
- Os vários tipos de *intents* permitem definir tráfego para diferentes abordagens no gerenciamento da rede.
- Além dos outros tipos de *intents* disponibilizados o *intent* um-para-muitos e o muitos-para-um, apresentam ser vantajosos na implementação de *broadcast* de áudio pela rede.

A rede SDN ainda apresenta alguma interoperabilidade entre dispositivos, o protocolo OpenFlow, e os controladores, como por exemplo a configuração *setQueue* suportada pelo *intent* no ONOS, suportada pelo OpenFlow, suportada pelos Open vSwitches, não é suportada pelos comutadores HP Aruba.

Em relação ao controlador ONOS, foram identificados as seguintes limitações:

- Apesar de suportar a configuração de regras de fluxos com *setQueue*, o ONOS não suporta a configuração automática das *queues* nos comutadores da rede. O que se torna um ponto fraco, pois as definições das *queues* têm de ser feitas manualmente nos comutadores de forma individual, como foi o caso deste projeto.
- O ONOS permite a configuração do *meters tables* nos dispositivos da rede, através do seu REST API e CLI, usando o protocolo OpenFlow na implementação de regras que encaminham o tráfego para a tabela correta. Contudo o *intent framework* não suporta, não possibilitando assim o benefício do *intent* para a configuração geral da rede.
- O controlador ONOS não disponibiliza uma aplicação que tenha como finalidade a implementação da qualidade de serviço. Contudo disponibiliza meios para o desenvolvimento.

## 5.1 Trabalho Futuro

A realização deste projeto foi feita, baseando-se na exploração das capacidades dos dispositivos e *software* disponibilizados, e na procura de uma solução de qualidade de serviço utilizando as suas capacidades provenientes de fabrico, sem criar nenhum modelo novo de QoS.

Assim sendo, como trabalho futuro e melhoria nas diferentes capacidades exploradas, destacam-se as seguintes. No contexto utilizado para implementar a qualidade de serviço, explorar melhor as *queues* nos Open vSwitches, uma vez que estes têm um campo DSCP. Com o objetivo de se aproximar cada vez mais do modelo *differentiated service* implementado nos comutadores físicos da HP.

Outro ponto que podia ser aprofundado é o *intent* e a implementação das suas configurações que permitem proporcionar a qualidade de serviço desejada. Principalmente em relação as suas opções de latência e largura de banda já disponíveis no CLI do ONOS, porém sem funcionalidade até o momento.

Ainda na topologia implementada, tem-se como trabalho futuro a inclusão de um outro comutador de fornecedor diferente aos implementados até o momento, juntamente com a inclusão de pontos de acesso. De modo a alargar a qualidade de serviço até esses dispositivos e testar a sua performance numa rede mais complexa e diversificada.

Tem-se ainda como trabalho futuro investigar um outro controlador para se adicionar a rede. De modo a fazer um estudo comparativo com o ONOS em relação a rede implementada. Tendo como referência as limitações encontradas e os principais benefícios do ONOS.

Relativamente a rede híbrida, quando se fez a configuração das duas redes, conseguiu-se a ligação direta entre as duas. Porém ainda se tem por realizar alguns testes nas configurações das bridges e no modo de comunicação com o controlador. Uma vez que as bridges permitem muitas configurações, como por exemplo a configuração de uma *fake bridge*, o que pode conduzir ao surgimento de uma comunicação *out-of-band* da rede virtual.



# Bibliografia

- [1] AES. *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*. Audio Engineering Society, Inc. Dezembro de 2016. Dez. de 2013. URL: <https://www.aes.org/standards/comments/drafts/aes67-r-171107-draft-rev-cfc.pdf> (ver pp. 31, 34).
- [2] Sugeng Winarno; Istiyanto Jazi Eko; Mustofa Khabib; Ashari Ahmad. «The Impact of QoS Changes towards Network Performance». English. Em: *International Journal of Computer Networks and Communications Security* 3 (2015), pp. 48–53. ISSN: 2308-9830 (Online) (ver pp. 32–33).
- [3] ARCHWAVE. *Connecting audio*. Online. Jan. de 2021. URL: <https://www.archwave.net/> (ver p. 1).
- [4] Audinate. *Dante Audio Networking*. Online. Acedido 20 Julho 2021. URL: <https://www.audinate.com/> (ver p. 30).
- [5] Universidade Federal da Bahia. «Conceitos básicos e visão geral de OpenFlow/SDN». Em: *Oficina de OpenFlow/SDN Universidade Federal da Bahia* (jul. de 2017), pp. 1–67 (ver pp. 19, 21).
- [6] Remo Balaguer. *Flow Embedding Algorithms for Software Defined Audio Networks*. 2015 (ver pp. 30–31, 36).
- [7] Sakir Sezer; Sandra Scott-Hayward; Pushpinder Kaur Chouhan. «Are We Ready for SDN? Implementation Challenges for Software-defined networks». Em: *IEEE Communications Magazine* (jun. de 2013) (ver p. 6).
- [8] CobraNet. *Networked digital audio*. Online. Acedido 15 Julho 2021. URL: <https://www.cobranet.info/> (ver p. 30).
- [9] Minh Pham Doan B Hoang. «SDN applications - the intent-based Northbound interface realisation for extended applications». Em: *iNEXT: Centre for Innovation in IT Services and applications* (dez. de 2016) (ver p. 13).
- [10] João Victor Escovar. *Tráfego na Internet vira mais uma preocupação para o país*. Online. Jan. de 2021. URL: <https://www.oconsumerista.com.br/2020/03/trafego-internet-preocupacao-pais/> (ver p. 1).
- [11] Open Networking Foundation. *MININET*. Online. Dez. de 2020. URL: <https://opennetworking.org/mininet/> (ver pp. 24–25, 39).

- [12] Open Networking Foundation. *ONOS - Open Network Operating System*. Online. Jan. de 2021. URL: <https://opennetworking.org/onos/> (ver p. 40).
- [13] Open Networking Foundation. *OpenFlow Switch Specification*. Online. Version 1.3.0. Jun. de 2012. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf> (ver p. 24).
- [14] Open Networking Foundation. *Software-Defined Networking (SDN) Definition*. Online. Jan. de 2021. URL: <https://opennetworking.org/sdn-definition/> (ver p. 5).
- [15] Open Networking Foundation. «Software-Defined Networking: The New Norm for Networks». English. Em: *ONF White Paper* (abr. de 2012), pp. 1–12 (ver p. 17).
- [16] Ronak Al-Haddad e Erika Sanchez Velazquez. «A Survey of Quality of Service Protocols and Software-Defined Networks - From the Traditional to the Latest Network Architecture». Em: *Springer Nature Switzerland AG* (jan. de 2019) (ver pp. 4, 33).
- [17] IBM. *What is Software-Defined Networking (SDN)?, SDN Versus Traditional Networking Explained*. Online. Out. de 2021. URL: <https://www.ibm.com/services/network/sdn-versus-traditional-networking> (ver p. 3).
- [18] Aruba - Hewlett Packard Enterprise Development LP. *Aruba 2930F / 2930M Advanced Traffic Management Guide for ArubaOS Switch 16.10*. Fev. de 2020. URL: [https://support.hpe.com/hpesc/public/docDisplay?docId=a00091305en\\_us](https://support.hpe.com/hpesc/public/docDisplay?docId=a00091305en_us) (ver p. 54).
- [19] Aruba - Hewlett Packard Enterprise Development LP. *ARUBA 2930M SWITCH SERIES*. Fev. de 2021. URL: [https://www.arubanetworks.com/assets/ds/DS\\_2930MSwitchSeries.pdf](https://www.arubanetworks.com/assets/ds/DS_2930MSwitchSeries.pdf) (ver p. 40).
- [20] Aruba - Hewlett Packard Enterprise Development LP. *Aruba 2930M Switches Installation and Getting Started Guide*. Fev. de 2020. URL: [https://support.hpe.com/hpesc/public/docDisplay?docId=a00015161en\\_us&docLocale=en\\_US](https://support.hpe.com/hpesc/public/docDisplay?docId=a00015161en_us&docLocale=en_US) (ver p. 40).
- [21] Aruba - Hewlett Packard Enterprise Development LP. *HPE ArubaOS-Switch OpenFlow v1.3 Administrator Guide for 16.03*. Mai. de 2017. URL: [https://support.hpe.com/hpesc/public/docDisplay?docLocale=en\\_US&docId=emr\\_na-c05365339](https://support.hpe.com/hpesc/public/docDisplay?docLocale=en_US&docId=emr_na-c05365339) (ver p. 59).
- [22] Glaucio França Giesen; Francine Cássia de Oliveira. «Abordagem SDN uma mudança de paradigma na arquitetura de rede tradicional». Em: *Instituto Nacional de Telecomunicações* (fev. de 2017), pp. 1–5 (ver p. 3).
- [23] Mininet org. *Mininet Python API Reference Manual*. Fev. de 2021. URL: <http://mininet.org/api/annotated.html> (ver p. 25).
- [24] Gustavo Pantuza. *Grafos como uma primitiva do plano de controle para análise e gerenciamento de redes definidas por software*. Instituto de Ciências Exatas da Universidade Federal de Minas Gerais. Mar. de 2015. URL: <https://www.dcc.ufmg.br/pos/cursos/defesas/1824M.PDF> (ver p. 18).
- [25] A Linux Foundation Collaborative Project. *Open vSwitch*. Online. 2016. URL: <https://www.openvswitch.org/> (ver p. 25).

- [26] Kostas Pentikousis; Jarno Pinola; Esa Piri; Pedro Neves; Susana Sargento; *Advances in Computers*. Marvin Zelkowitz, 2010. ISBN: 9780123810205 (ver p. 30).
- [27] Nuno Tiago Louro Simões. *SDN: os novos desafios na gestão de redes de comunicação*. Insitituto politécnico de Leiria, mai. de 2016. URL: <http://hdl.handle.net/10400.8/1905> (ver p. 1).
- [28] Ashutosh Kumar Singh; Shashank Srivastava. «A Survey and classification of controller placement problem in SDN». Em: *International Journal of Network Management* (mar. de 2018), pp. 1–5 (ver p. 32).
- [29] TELEDYNAMICS. *The advantages of SDN for VoIP*. Online. Acedido 5 Agosto 2021. URL: <http://info.teledynamics.com/blog/the-advantages-of-sdn-for-voip> (ver p. 38).
- [30] Goransson Paul; Black Chuck; Culver Timothy. *Software Defined Networks - A Comprehensive Approach*. Todd Green, 2017. ISBN: 978-0-12-804555-8 (ver pp. 4–5, 7, 11, 21–22, 24).
- [31] Matteus Wolff. *O que SDN e como essa arquitetura de rede funciona*. 2011. URL: <https://penseemti.com.br/artigos/o-que-e-sdn-e-como-essa-arquitetura-de-rede-funciona/> (ver p. 3).
- [32] Clark Zoeller. *How Does QoS Work?* Packet Pushers. Dez. de 2013 (ver p. 37).



## Apêndice A

# Configurações comutadores HP Aruba

Controlador:

- IP: 192.168.100.3
- Máscara: 255.255.255.0

Comutador HP Aruba 1:

- IP OOBM: 192.168.100.5
- IP Vlan: 192.168.100.4
- Máscara: 255.255.255.0

Comutador HP Aruba 2:

- IP OOBM: 192.168.100.7
- IP Vlan: 192.168.100.6
- Máscara: 255.255.255.0

Comutador HP Aruba 3:

- IP OOBM: 192.168.100.1
- IP Vlan: 192.168.100.2
- Máscara: 255.255.255.0

NA figura [A.1](#) encontra-se apresentada as configurações da primeira instância OpenFlow 1.0 configurada nos comutadores HP Aruba.

```

Aruba-2930M-24G-PoEP# show openflow instance aggregate

Configured OF Version      : 1.0
Negotiated OF Version      : 1.0
Instance Name              : aggregate
Data-path Description       : aggregate
Administrator Status       : Enabled
Member List                : VLAN 1
Pipeline Model             : NA
Listen Port                : None
Operational Status         : Up
Operational Status Reason  : NA
Datapath ID                : 0001104f58fa0000
Mode                       : Active
Flow Location              : Hardware and Software
No. of Hardware Flows      : 3
No. of Software Flows      : 3
Hardware Rate Limit        : 0 kbps
Software Rate Limit        : 100 pps
Conn. Interrupt Mode       : Fail-Secure
Maximum Backoff Interval   : 60 seconds
Probe Interval             : 10 seconds
Hardware Table Miss Count  : 1418219
No. of Software Flow Tables : NA
Egress Only Ports         : None
Table Model               : Single Table
Source MAC Group Table     : Disabled
Destination MAC Group Table : Disabled
Default Miss Action        : Drop
Packet-In VLAN Tagging    : Default

Controller Id Connection Status Connection State Secure Role
-----
1           Connected       Active           No       Equal

```

Figura A.1: Instância aggregate inicial

```

Aruba-2930M-24G-PoEP# show openflow instance aggregate flows

OpenFlow Flow Table

Flow 1
Match
  Incoming Port : Any
  Source MAC    : Any
  Destination MAC Mask : 000000-000000
  VLAN ID      : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  ARP Opcode   : Any
  Source Port  : Any
  Ethernet Type : ARP
  Destination MAC : Any
  VLAN Priority  : Any
  IP ToS Bits   : Any
  Destination Port : Any
Attributes
  Priority      : 40000
  Hard Timeout : 0 seconds
  Byte Count   : 0
  Controller ID : 1
  Flow Location : Hardware
  Hardware Index : 0
  Reason Code   : 12
  Reason Description : Rule is in hardware.
Actions
  Controller Port
  Duration      : 18295 seconds
  Idle Timeout  : 0 seconds
  Packet Count  : 38
  Cookie        : 0x10000e0518273

```

Figura A.2: Regra de fluxo do comutador HP para Openflow 1.0

```
Flow 5
Match
Incoming Port : 10
Source MAC : Any
Source MAC Mask : 000000-000000
Destination MAC Mask : 000000-000000
VLAN ID : Any
Source IP Address : Any
Destination IP Address : Any
IP Protocol : Any
IP ECN : Any
Source Port : Any
Source Port Range : NA
Destination Port Range : NA
TCP Flags : NA
TCP Mask : NA
Ethernet Type : Any
Destination MAC : Any
VLAN Priority : Any
IP DSCP : Any
Destination Port : Any
Custom Match One : Any
Custom Match One Mask : Any
Custom Match Two : Any
Custom Match Two Mask : Any
Custom Match Three : Any
Custom Match Three Mask : Any
Custom Match Four : Any
Custom Match Four Mask : Any
Attributes
Priority : 100
Hard Timeout : 0 seconds
Byte Count : NA
Flow Table ID : 0
Cookie : 0xbe000034b1afb1
Hardware Index : 0
Duration : 510373 seconds
Idle Timeout : 0 seconds
Packet Count : 5869
Controller ID : 1
Instructions
Apply Actions
Output : 9
```

Figura A.3: Regra de fluxo do comutador HP para Openflow 1.3



## Apêndice B

### Scripts para emulação da rede

```

!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf, OVSLink

def myNetwork():
    net = Mininet( topo=None,
                  build=False,
                  ipBase='192.168.100.0/24')

    c0=net.addController( name='c0',
                          controller=RemoteController,
                          ip='192.168.100.3',
                          protocol='tcp',
                          port=6633)

    info('Add Switches.....')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, protocols=["OpenFlow13"])
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, protocols=["OpenFlow13"])
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, protocols=["OpenFlow13"])

    info('Add Hosts.....')
    h1 = net.addHost('h1', cls=Host, ip='192.168.100.13', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='192.168.100.14', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='192.168.100.15', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='192.168.100.16', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='192.168.100.17', defaultRoute=None)

    info('Add Link.....')
    net.addLink(s1, s2)
    net.addLink(s1, s3)
    net.addLink(s2, s3)
    net.addLink(s1, h1)
    net.addLink(s2, h2)
    net.addLink(h3, s2)
    net.addLink(h4, s3)
    net.addLink(h5, s3)

    net.build()

    net.start()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()

```

Bloco de Código B.1: Script de emulação da rede em Python – Mininet

```

class VLANHost(Host):

    def config(self, vlan=100, **params):

        r = super(VLANHost, self).config(**params)

        intf = self.defaultIntf()

        self.cmd('ifconfig %s inet 0' % intf)

        self.cmd('vconfig add %s %d' % (intf, vlan))

        self.cmd('ifconfig %s.%d inet %s' % (intf, vlan, params['ip']))

        newName = '%s.%d' % (intf, vlan)

        intf.name = newName

        self.nameToIntf[newName] = intf

    return r

```

Bloco de Código B.2: Script de emulação da rede em Python – Mininet – Host enviando tags

```

echo "Congifurando fake-bridge switch s10 e Adicionando interface enpls0 a
    brigde...."

sudo ovs-vsctl add-br s10 s1 100
sudo ovs-vsctl add-port s10 enpls0
sudo ifconfig enpls0 0
sudo ifconfig s10 192.168.100.11 netmask 255.255.255.0
sudo route add default gw 192.168.100.1 s10
sudo ip link set s10 up

sudo ovs-vsctl set interface enpls0 other_config:qinq-ethertype=802.1q

```

Bloco de Código B.3: Script de configuração da ligação – com interface física em Vlan

```
echo "Congifurando bridge switch s1 e Adicionando interface enpls0 a brigde...."
sudo ovs-vsctl add-port s1 enpls0
sudo ifconfig enpls0 0
sudo ifconfig s1 192.168.100.11 netmask 255.255.255.0
sudo route add default gw 192.168.100.1 s1
sudo ip link set s1 up

echo "Congifurando as bridges das switches para funcionamento in-band..."
sudo ovs-vsctl set bridge s1 other_config=disable-in-band=false
sudo ovs-vsctl set bridge s2 other_config=disable-in-band=false
sudo ovs-vsctl set bridge s3 other_config=disable-in-band=false

echo "Configurando connection-mode de cada switch para in-band.... "
CONTROLLER=$(sudo ovs-vsctl list controller | grep _uuid | awk '{print $3;}')

for arg in $CONTROLLER
do
    ovs-vsctl set controller $arg connection-mode=in-band
    ovs-vsctl set controller $arg other_config:dscp=48
done
```

Bloco de Código B.4: Script de configuração da ligação da rede virtual com rede física

```

ovs-vsctl -- set port enpls0 qos=@AES67qos -- --id=@AES67qos create qos
    type=linux-htb queues=1=@q1,2=@q2 --\
    --id=@q1 create queue other-config:min-rate=20000
        other-config:max-rate=100000000 --\
    --id=@q2 create queue other-config:min-rate=10000000
        other-config:max-rate=100000000

ovs-vsctl -- set port s1-eth2 qos=@qos1 -- --id=@qos1 create qos type=linux-htb
    queues=1=@q1,2=@q2 --\
    --id=@q1 create queue other-config:min-rate=20000
        other-config:max-rate=100000000 --\
    --id=@q2 create queue other-config:min-rate=10000000
        other-config:max-rate=100000000

ovs-vsctl -- set port s3-eth1 qos=@qos2 -- --id=@qos2 create qos type=linux-htb
    queues=1=@q1,2=@q2 --\
    --id=@q1 create queue other-config:min-rate=20000
        other-config:max-rate=100000000 --\
    --id=@q2 create queue other-config:min-rate=10000000
        other-config:max-rate=100000000

ovs-vsctl -- set port s1-eth3 qos=@qos1 -- --id=@qos1 create qos type=linux-htb
    queues=1=@q1,2=@q2 --\
    --id=@q1 create queue other-config:min-rate=20000
        other-config:max-rate=100000000 --\
    --id=@q2 create queue other-config:min-rate=10000000
        other-config:max-rate=100000000

ovs-vsctl -- set port s3-eth4 qos=@qos2 -- --id=@qos2 create qos type=linux-htb
    queues=1=@q1,2=@q2 --\
    --id=@q1 create queue other-config:min-rate=20000
        other-config:max-rate=100000000 --\
    --id=@q2 create queue other-config:min-rate=10000000
        other-config:max-rate=100000000

```

Bloco de Código B.5: Script queueConfig