Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2013

Rafael Gouveia

Demonstrador de uma rede *Openflow*

"As dificuldades preparam as pessoas comuns para destinos extraordinários"

— C.S.Lewis

Rafael
Gouveia

Demonstrador de uma Rede *Openflow*

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica de Susana Sargento, Professora Doutora Auxiliar do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro, e co-orientação científica do Mestre Jorge Carapinha, Engenheiro do Departamento de Coordenação Tecnológica e Inovação Exploratória da Portugal Telecom Inovação.

o júri / the jury

presidente / president

Rui Luís Andrade Aguiar

Professor Associado C/ Agregacao, Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

vogais / examiners committee

Susana Isabel Barreto de Miranda Sargento

Professora Doutora Auxiliar do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro(orientador)

Jorge Manuel dos Santos Correia Carapinha

Investigador Sénior da Portugal Telecom Inovação (co-orientador)

Paulo Alexandre Ferreira Simões

Professor Auxiliar, Dep. de Engenharia Informática da Fac. de Ciências e Tecnologia da Universidade de Coimbra

agradecimentos / acknowledgements

Primeiro que tudo queria agradecer a Deus por me ter permitido chegar até aqui, sem ele nada disto teria sido possível. A sua força e apoio foram uma mais-valia ao longo deste longo e dificultoso caminho, pois guiou-me colocando no meu caminho as pessoas correctas e adequadas nos momentos certos, entenda-se de maior dificuldade. Por isso um Muito Obrigado fica curto para a graça que Deus me concedeu.

Gostaria de agradecer à Prof. Dra. Susana Sargento por me ter aceitado no momento da escolha da tese, e ter-me permitido ter crescido profissional e pessoalmente ao longo desta Dissertação pelos desafios propostos e apoio que apresentou.

Ao meu co-orientador o Engenheiro/Mestre Jorge Carapinha por todo o apoio, conceitos, disponibilidade e interesse que sempre demonstrou ao longo do desenvolvimento do meu trabalho.

Ao Mestre Márcio Melo pelo seu apoio na familiarização dos conceitos relacionados com este trabalho científico.

Aos Mestres João Soares e Bruno Parreira que foram os pilares fundamentais do desenvolvimento do meu trabalho e sem os quais tudo se teria tornado muito mais difícil do que já foi. Agradeço a vossa paciência para a minha ignorância e pouca experiência, todos os conhecimentos, dicas e palavras de alento de atenção que me foram dando ao decorrer da Dissertação que permitiram evoluir e crescer em todos os aspectos da minha vida.

A nível pessoal queria agradecer a todos os meus professores, em especial, ao Professor Gonçalo Amorin que foi um guia para mim no momento da escolha deste curso. Também ao Professor Arnaldo Oliveira, Carlos Brites e Lisa Brites que foram um apoio e guias incondicionais neste percurso académico e que sem eles talvez hoje não estivesse cá. Por outro lado aos meus colegas e amigos que sempre me acompanharam neste percurso, em particular a Andreína Teixeira da França, Aurélio Câmara, Dinarte Silva, Bruno Gomes, Dina Trindade, Yodercy Fernandes, Filipe Rodrigues e a todos os integrantes do grupo "pessoal do tacho"que mesmo não estando perto fisicamente souberam animar-me e manter-me focado e animado nos piores momentos para conseguir atingir este meu objectivo. Um muito Obrigado

Por outro lado à minha mãe e meu irmão pela paciência e compreensão que tiveram para comigo durante todos estes anos e por me terem dado tudo seu apoio que me permitiram chegar a ser a pessoa que sou hoje.

Finalmente, ao Instituto de Telecomunicações de Aveiro e seus colaboradores por as condições e apoios fornecidos para o desenvolvimento desta tese.

Resumo

A Internet constitui hoje uma das infra-estruturas críticas da nossa sociedade, pois fornece uma série de serviços para todas as faixas etárias. Contudo a sua arquitectura e princípio de funcionamento foram desenvolvidos para redes simples e de pequenas dimensões que suportavam um pequeno conjunto de serviços e aplicações. O problema surge quando a utilização da Internet ultrapassa os limites para as quais foi desenvolvida. Factores como um grande aumento do número de utilizadores, multiplicidade de dispositivos e serviços levaram à necessidade de criar protocolos e mecanismos para colmatar os problemas induzidos pelos mesmos. Estes últimos acabaram por tornar um sistema simples para troca de informação num sistema complexo, pouco escalável e flexível, tornando cada vez mais difícil a inclusão de novas tecnologias e protocolos, o que acaba por limitar a inovação e a introdução de novos serviços com suporte na rede.

Estes problemas motivaram a comunidade científica à procura de novas soluções e ideias para uma arquitectura que satisfaça os requisitos actuais do mercado. Foi neste contexto que surgiram as *Software Defined Networks (SDN)* .

As *SDNs* constituem um novo paradigma de redes de comunicações que visa dar resposta aos problemas anteriormente referidos. Por outro lado é uma tecnologia jovem e em franca expansão e desenvolvimento. É neste âmbito que surge este projecto de Dissertação, cujo objectivo é realizar um demonstrador de rede *Openflow* que permita o estudo e análise das suas capacidades em condições similares às encontradas num cenário real. Também se pretende que esta possua um mecanismo de instanciação e de reconfiguração inteligente da rede.

Com este objectivo em mente foi realizado um estudo aprofundado sobre as *SDNs*. Nesse estudo são abordados alguns dos conceitos e tecnologias relacionados com a mesma, como por exemplo, possíveis plataformas e protocolos de utilização que promovam o aumento das suas capacidades e funcionalidades. Neste âmbito foi desenvolvido um demonstrador de redes *Openflow* baseado na arquitectura das SDNs com o objectivo de verificar o resultado do estudo teórico realizado sobre as mesmas. O demonstrador terá um papel preponderante na camada de controlo, tendo sido necessário o desenvolvimento do gestor de rede responsável pelo controlo da rede *Openflow*. Este gestor será constituído por 4 módulos, dos quais dois são alvo de estudo e desenvolvimento neste projecto de dissertação. O primeiro destes dois módulos é o responsável pela monitorização da rede, isto é, verificar o seu estado e notificar possíveis problemas da mesma. O segundo módulo será o responsável pela activação dos fluxos correspondentes aos serviços que se pretendem activar na rede. Relativamente aos outros dois módulos podemos referir que correspondem a módulos externos cujo objectivo de integração no demonstrador é permitir testar as funcionalidades do demonstrador *Openflow*. Finalmente foram desenvolvidos mecanismos que garantem a comunicação entre os módulos.

A fim de validar as premissas que deram origem a este projecto de investigação, foram efectuados um conjunto de testes sobre as funcionalidades e desempenho de cada um dos módulos desenvolvidos.

Abstract

Nowadays, Internet is one of the most critical infrastructures of our society, as it provides a wide range of services for several age groups. However, its architecture and operating principles were developed for simple and small-sized networks that only supported a few set of services and applications. The problem arises when Internet usage exceeds the limits for which it was previously designed. Reasons such as the large increase of the number of users, multiplicity/heterogeneity of devices and services, led to the need of creating protocols and mechanisms to solve the problems. The latter is responsible for making a rather simple data exchange system into a complex one, poorly scalable and flexible. It has become more difficult to include new technologies and protocols, restraining innovation and the introduction of new services supported in the network.

The above mentioned issues motivated the scientific community to look for novel solutions and ideas towards an architecture that fulfills the current market requirements. Software Define Networking emerged in this context.

SDN appears as a new paradigm of communication networks that aims to answer the problems earlier stated. Still, this is a novel technology that is in fairly expansion and development. This Dissertation project arises in that scope, whose purpose is to create a setup that allows studying and analyzing its capacity in a real case scenario. Furthermore, the setup also contains a mechanism for instantiation and intelligent reconfiguration of the network.

A depth research on *SDNs* was made. This research addresses concepts and technologies such as possible platforms and utilization protocols that stimulate the increase of capacity and functionality of *SDN*s.Under this heading, an openflow network setup based on *SDN* architecture was developed, with the aim of verifying the outcomes of the theoretical study previously done. The setup plays an important role on the control layer, and it has been developed a network management module responsible for the Openflow network control. This management part is composed by 4 distinct blocks, 2 of them are targeted and built in this Dissertation project. The first of them is the one responsible for the network monitoring, more precisely, it requires verifying the general order of the network and notify possible problems that might occur. The second one is responsible for the streams activation of the corresponding services that are enabled in the network. Regarding the other two modules, they represent external modules whose integration in the setup aims to allow testing the functionalities of the Openflow setup itself. Finally, mechanisms that guarantee communication between modules were developed.

In order to validate the assumptions that gave rise to this investigation project, a set of tests were carried out, related to functionality and performance of each module built. Finally, a comparison between the performance of this implementation and the performance of current networks was held.

Índice

Índice i

Lista de Figuras v

Lista de Tabelas ix

1 Introdução 1

1.1 Motivação . . . 1

1.1.1 Arquitectura actual da Internet . . . 1

1.1.2 *SDN*, uma arquitectura emergente . . . 2

1.2 Proposta . . . 3

1.3 Organização da tese . . . 4

2 Estado da Arte 5

2.1 Introdução . . . 5

2.2 *Software Defined Networks (SDN)* . . . 5

2.2.1 Arquitectura . . . 7

2.2.2 Redes Actuais vs *SDN* . . . 8

2.2.3 Vantagens e Desvantagens das *SDNs* . . . 9

2.3 Virtualização de rede . . . 10

2.3.1 *SDNs* e Virtualização de rede . . . 11

2.4 *Cloud Computing (CC)* . . . 12

2.4.1 *Cloud Computing (CC)* e *SDNs* . . . 12

2.5 Protocolos e Tecnologias . . . 13

2.6 *Openflow* . . . 14

2.6.1 *Openflow Protocol* . . . 14

2.6.2 *Openflow Switch* . . . 16

2.6.3 *Openflow Controller* . . . 16

2.6.3.1 *NOX* . . . 17

2.6.3.2 *POX* . . . 18
2.6.3.3 *Simple Network Access Control (SNAC)* . . . 18
2.6.3.4 *Beacon* . . . 18
2.6.3.5 Maestro . . . 19
2.6.3.6 *FlowVisor* . . . 19
2.6.3.7 *Floodlight* . . . 20
2.6.3.8 *Frenetic* . . . 20
2.6.3.9 Comparação entre os controladores . . . 21
2.6.4 *Openflow Channel* . . . 21
2.7 Projectos e Plataformas . . . 22
2.7.1 *Nicira* . . . 23
2.7.2 *Hewlett-Packard (HP)* . . . 24
2.7.3 *Cisco* . . . 25
2.7.4 *OpenStack* . . . 25
2.7.5 Pertino . . . 26
2.8 Conclusão . . . 27

3 Arquitectura do Demonstrador de rede *Openflow* 29
3.1 Introdução . . . 29
3.2 Proposta de solução . . . 29
3.2.1 Descrição da solução . . . 29
3.2.2 Arquitectura da solução . . . 30
3.3 Gestor de Rede . . . 31
3.3.1 Arquitectura e funcionalidades . . . 31
3.3.1.1 *Service Manager* . . . 31
3.3.1.2 *Monitoring* . . . 32
3.3.1.3 *Activator* . . . 33
3.3.1.4 *Openflow Controller* . . . 33
3.4 Rede *Openflow* . . . 35
3.5 Comunicação entre módulos . . . 37
3.5.1 *Representational State Transfer (REST)* . . . 37
3.5.2 *Java Script Object Notation (JavaScript Object Notation (JSON))* . . . 38
3.5.3 Comunicação entre o *Monitoring-Floodlight* . . . 39
3.5.4 Comunicação entre o *Service Manager-Monitoring* . . . 39
3.5.5 Comunicação entre o *Service Manager-Activator* . . . 39
3.5.6 Comunicação entre o *Activator-Floodlight* . . . 39

3.5.7 Comunicação entre o *Floodlight* e a rede *Openflow* 40
3.6 Requisitos e Dependências de *Software* 41
3.7 Conclusão 41

4 Implementação do Demonstrador *Openflow* 43
4.1 Introdução 43
4.2 *Testbed* 43
4.3 *Monitoring* 45
4.3.1 Arquitectura 45
4.3.2 Funcionalidades e Algoritmos 46
4.3.2.1 Serviço de Topologia 46
4.3.2.2 Subscrição ao serviço de alarme 48
4.3.2.3 Alarme Perda de *Link* 49
4.3.2.4 Alarme percentagem de perdas 49
4.3.2.5 Fornecimento de informação estatística 50
4.3.3 *Uniform Resource Identifier (URL) s* 50
4.4 *Activator* 51
4.4.1 Funcionalidades e Algoritmos 51
4.4.1.1 Activação de um fluxo 51
4.4.1.2 Actualização de um fluxo 53
4.4.1.3 Eliminação de um Serviço 54
4.4.2 *URLs* disponibilizados pelo módulo Activator 54
4.5 Conclusão 55

5 Testes e Análise dos Resultados 57
5.1 Introdução 57
5.2 Análise ao módulo *Monitoring* 57
5.2.1 *Testbed* 57
5.2.2 Perda de *Link* 57
5.2.2.1 Metodologia 58
5.2.2.2 Resultados 58
5.2.3 Detecção de um novo *Link* 58
5.2.3.1 Metodologia 58
5.2.3.2 Resultados 58
5.2.3.3 Conclusão 59
5.3 Análise ao módulo Activator 59
5.3.1 *Testbed* 59

5.3.2 Metodologia . . . 63
5.3.3 Teste de activação do serviço *Address Resolution Protocol (ARP)* . . . 63
5.3.4 Teste de activação do serviço *Transmission Control Protocol (TCP)* . . 64
5.3.4.1 Topologia de 4 nós . . . 64
5.3.4.2 Topologia de 8 nós . . . 65
5.3.4.3 Topologia de 16 nós . . . 65
5.3.4.4 Topologia de 32 nós . . . 67
5.3.4.5 Conclusão . . . 69
5.3.5 Teste de activação do serviço *User Datagram Protocol (UDP)* . . . 69
5.3.5.1 Topologia de 4 nós . . . 69
5.3.5.2 Topologia de 8 nós . . . 70
5.3.5.3 Topologia de 16 nós . . . 72
5.3.5.4 Topologia de 32 nós . . . 72
5.3.5.5 Conclusão . . . 75
5.3.6 Teste de activação do serviço *Internet Control Message Protocol (ICMP)* 75
5.3.6.1 Topologia de 4 nós . . . 75
5.3.6.2 Topologia de 8 nós . . . 76
5.3.6.3 Topologia de 16 nós . . . 76
5.3.6.4 Topologia de 32 nós . . . 78
5.3.6.5 Conclusão . . . 80
5.3.7 Resultados Sumários . . . 80
5.4 Teste à Arquitectura Geral da Solução . . . 81
5.4.1 *Testbed* . . . 81
5.4.2 Metodologia . . . 81
5.4.3 Activação de serviço entre o nó A e o nó D . . . 82
5.4.4 Activação de serviço entre o nó A e os nós B e D . . . 83
5.4.5 Activação de serviço entre o nó A e os nós B e D . . . 83
5.4.6 Resultados Sumários . . . 84
5.5 Conclusão . . . 85

6 Conclusões **87**

Bibliografia **89**

Lista de Figuras

2.1 Comparação entre a arquitectura dos computadores e a arquitectura das SDN [38] . . . 6
2.2 Arquitectura SDN [19] . . . 7
2.3 Arquitectura Actual vs Arquitectura SDN [46] . . . 9
2.4 Compatibilidade entre SDN e NFV [4] . . . 11
2.5 Funcionamento do *Openflow Protocol* [3] . . . 15
2.6 Funcionamento do *Openflow switch* [3] . . . 16
2.7 SDN Controller . . . 17
2.8 Nox [36] . . . 18
2.9 FlowVisor [61] . . . 19
2.10 floodlight1 [64] . . . 20
2.11 Open Day Light [30] . . . 23
2.12 Solução *Openflow* da *Nicira* [48] . . . 24
2.13 HP History [5] . . . 24
2.14 Solução *Openflow* da Cisco [43] . . . 25
2.15 Solução *Openflow* da Pertino [27] . . . 26

3.1 Arquitectura do Gestor de Redes . . . 30
3.2 Arquitectura Interna do Gestor de Redes . . . 31
3.3 Arquitectura Interna do módulo *Service Manager* . . . 32
3.4 Floodlight1 [64] . . . 34
3.5 Arquitectura interna de um OpenVswitch . . . 36

4.1 *Testbed* criada . . . 44
4.2 Localização dos módulos presentes na solução . . . 45
4.3 Arquitectura Interna do módulo *Monitoring* . . . 45
4.4 Fluxograma do serviço topologia . . . 46
4.5 Fluxograma da função topologia . . . 47

4.6 Fluxograma da subscrição do serviço Alarmes 48
4.7 Fluxograma da alarme perda de *link* 49
4.8 Fluxograma de alarme percentagem de perdas 50
4.9 Fluxograma da activação de um serviço 52
4.10 Fluxograma da actualização do serviço 53

5.1 Topologia física de uma rede *Openflow* constituída por 8 nós 60
5.2 Topologia física de uma rede *Openflow* constituída por 16 nós 61
5.3 Topologia física de uma rede *Openflow* constituída por 32 nós 62
5.4 Tempo para a Activação de um fluxo *TCP* no substrato de 4 nós 64
5.5 Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 4 nós 65
5.6 Tempo para a Activação de um fluxo *TCP* no substrato de 8 nós 66
5.7 Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 8 nós 66
5.8 Tempo para a Activação de um fluxo *TCP* no substrato de 16 nós 67
5.9 Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 16 nós 68
5.10 Tempo para a Activação de um fluxo *TCP* no substrato de 32 nós 68
5.11 Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 32 nós 69
5.12 Tempo para a Activação de um fluxo *UDP* no substrato de 4 nós 70
5.13 Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 4 nós 71
5.14 Tempo para a Activação de um fluxo *UDP* no substrato de 8 nós 71
5.15 Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 8 nós 72
5.16 Tempo para a Activação de um fluxo *UDP* no substrato de 16 nós 73
5.17 Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 16 nós 73
5.18 Tempo para a Activação de um fluxo *UDP* no substrato de 32 nós 74
5.19 Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 32 nós 74
5.20 Tempo para a Activação de um fluxo *ICMP* no substrato de 4 nós 75
5.21 Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 4 nós 76

5.22 Tempo para a Activação de um fluxo *ICMP* no substrato de 8 nós 77
5.23 Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 8 nós 77
5.24 Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 16 nós 78
5.25 Tempo para a Activação de um fluxo *ICMP* no substrato de 16 nós 78
5.26 Tempo para a Activação de um fluxo *ICMP* no substrato de 32 nós 79
5.27 Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 32 nós 79
5.28 Metodologia utilizada para análise da arquitectura geral 81

Lista de Tabelas

2.1 Tabela de *Comparação entre os controladores* 22

3.1 Tabela dos *URLs* do *Service Manager* 33
3.2 Tabela dos URLs do *Floodlight* 35
3.3 Mensagens entre o *Floodlight* e a rede *Openflow* 40
3.4 Mensagens entre o *Floodlight* e a rede *Openflow* (continuação) 40

4.1 Especificação das características do *hardware* e *software* da *testbed* 44
4.2 URLs do módulo *Monitoring* 51
4.3 URLs disponibilizados pelo módulo *Activator* 55

5.1 Tabela dos resultados obtidos para a simulação da perda de um *link* na rede . . 58
5.2 Tabela dos resultados obtidos para a simulação da obtenção de um novo *link* na rede 59
5.3 Tabela dos resultados obtidos para a activação de um serviço *ARP* na rede . . 63
5.4 Tabela dos resultados obtidos para a simulação da perda de um *link* na rede . . 80
5.5 A activação de um serviço *UDP* que é decomposto em 1 fluxo 82
5.6 Activação de um serviço *UDP* que é decomposto em 2 fluxos 83
5.7 Activação de um serviço *UDP* que é decomposto em 19 fluxos 84
5.8 Tabela dos resultados globais para os vários serviços a activar na rede 84

Lista de Acrónimos

ARP	Address Resolution Protocol
API	Application Programming Interface
ARPA	Advanced Research Projects Agency
ARPANET	Advanced Research Projects Agency Network
CC	Cloud Computing
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DC	DataCenter
GSMP	Global Standards Management Process
HP	Hewlett-Packard
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMC	Intelligent Management Controller
IP	Internet Protocol
JSON	JavaScript Object Notation
LACP	Lateral Assessment Certification Program
L2	Layer 2
LAN	Local Area Network
LISP	LISt Processing
MAC	Media Access Control
MPLS	Multi Protocol Label Switching
NAAS	Network as a Service
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
NIST	National Institute of Standards and Technology
ONF	Open Networking Foundation
ONE	Open Network Environment
OPENSIG	Open Signaling
OSI	Open Systems Interconnection
OSGI	Open Services Gateway Initiative
PC	Personal Computer
PCAP	Packet Capture

QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
SDN	Software Defined Networks
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SNAC	Simple Network Access Control
SSL	Secure Sockets Layer
TELNET	Telecomunication Network
TCP	Transmission Control Protocol
URL	Uniform Resource Identifier
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VXLAN	Virtual Extensible LAN
XML	eXtensible Markup Language

Capítulo 1

Introdução

1.1 Motivação

1.1.1 Arquitectura actual da Internet

Desde meados dos anos 60 temos assistido a uma série de avanços tecnológicos e científicos no mundo das comunicações que se reflectiram na sociedade e que permitiram simplificar alguma das tarefas do nosso dia-a-dia.

Entre essas descobertas encontramos um dos pilares da Internet que conhecemos hoje, a *Advanced Research Projects Agency Network (ARPANET)* . O seu desenvolvimento teve origem durantes os anos 60 nos Estados Unidos da America, promovido pelo então *Advanced Research Projects Agency (ARPA)* hoje conhecido por *Defense Advanced Research Projects Agency (DARPA)* , no momento da Guerra Fria. O *ARPANET* visava então facilitar a troca de informação entre os investigadores científicos e militares de diferentes localizações [35] [33].

O sucesso alcançado pelo *ARPANET* e o desenvolvimento de serviços como o *Telecomunication Network (TELNET)* para a troca de informação motivou a comunidade científica a continuar a estudar a expansão destes serviços em áreas maiores. Esta expansão traduziu-se num aumento do número de utilizadores que provocou a saturação dos meios físicos e lógicos concebidos no projecto inicial para a comunicação. Mais tarde, após muitos outros desenvolvimentos também eles fundamentais, entre eles o protocolo *TCP*, surge a Internet actual. A sua arquitectura base é suportada por um modelo conceptual de camadas. Esta arquitectura é baseada nos princípios de um núcleo de redes simples e transparente, deslocando a inteligência para os sistemas finais que têm mais funcionalidades. Além disso é também uma arquitectura descentralizada e que se encontra dividida em múltiplas regiões administrativas. Estes princípios respondiam perfeitamente às necessidades das redes militares e universitárias, pois permitiam a criação de uma rede de larga escala composta por diferentes redes com entidades administrativas próprias. Contudo quando foi disponibilizada à população em geral, em início da década de 90, uma pequena aldeia transformou-se numa das principais capitais mundiais, dada a quantidade de informação gerada e partilhada pelos utilizadores. Esta transformação obrigou à integração de uma série de novas tecnologias e protocolos a fim de fornecer maior mobilidade, qualidade de serviço e segurança a fim de responder às necessidades do mercado existente [42] [33].

Actualmente, com a explosão de dispositivos móveis e aplicações que recorrem à Internet surge também um conjunto de problemas a nível da arquitectura actual da Internet tais como:

- Ossificação da Internet: a integração ou fornecimento de um novo serviço de rede obriga à inclusão de novos protocolos de comunicação, transporte e rede que garantem o correcto funcionamento da mesma independente do utilizador final. Este problema surge devido à arquitectura estática e simplista que tinha como objectivo o estabelecimento de comunicação entre estações, e que a fim de colmatar as necessidades actuais do mercado tem recorrido a uma pilha complexa de protocolos.
- Falta de Escalabilidade: a escalabilidade constitui um dos maiores problemas da arquitectura actual, pois o crescente número de utilizadores leva a um aumento exponencial das tabelas de *routing*, que se traduz num aumento da complexidade a inserir na rede para garantir a comunicação entre dispositivos e o correcto funcionamento dos serviços fornecidos pela Internet [19].
- Mobilidade: com o aumento dos dispositivos móveis, a garantia de acesso à Internet em diferentes pontos de acesso vê-se prejudicada, pois vai contra os pressupostos da arquitectura actual que se baseia num modelo de comunicação cliente-servidor, ou seja, em que grande parte do tráfego e da comunicação passa pelo núcleo da rede [19].
- Alteração dos padrões de tráfego: o crescente número de novos dispositivos ligados à rede tem vindo a provocar alterações nos padrões e rotas de tráfego. Este aumento prejudica claramente a eficiência dos algoritmos utilizados para o mapeamento da rede, já que estes foram desenvolvidos para uma rede de pequena dimensão onde a gestão estática era preferível [19].
- Garantia de *Quality of Service (QoS)* : o aumento dos requisitos de serviços e aplicações na Internet, juntamente com o aumento de utilizadores na Internet, tem comprometido a garantia de *QoS* dos mesmos. Isto acontece porque a actual arquitectura de rede apenas oferece um serviço de *best-effort*. Contudo não oferece garantias relativamente ao controlo de admissão, não tem políticas nem uma metodologia de redistribuição de tráfego a fim de manter a qualidade de serviço para todas as aplicações que pretendem ser atendidas. Actualmente, a escolha dos operadores para garantir *QoS* baseia-se no sobre-aprovisionamento de recursos o que constitui um grande desperdício dos mesmos [42][24][19].
- Integração de novas tecnologias e paradigmas: a actual arquitectura estática das redes apresenta bastantes dificuldades em acompanhar o surgimento de novas tecnologias e paradigmas, como por exemplo o *Cloud Computing* (*CC*), acabando por dificultar a disseminação e o bom funcionamento das mesmas. A maioria dos dispositivos de redes apresentam sistemas operativos proprietários que obrigam ao desenvolvimento de novos protocolos e ferramentas sempre que se quer integrar uma nova tecnologia [19][39][63].

Como podemos observar, a rede actual apresenta um conjunto de limitações devido à concepção inicial do seu projecto, impedindo-a assim de acompanhar as exigências e tendências dos utilizadores de hoje em dia, o que tem provocado a perda de qualidade no serviço prestado.

1.1.2 *SDN*, uma arquitectura emergente

Tendo em conta os problemas da arquitectura actual das redes de comunicação, a comunidade científica tem vindo a trabalhar em alternativas. Uma das mais proeminentes baseia-se num novo tipo de arquitectura de rede, mais flexível, versátil e de maior abstracção, que utiliza conceitos de virtualização e permite a programação das mesmas, a fim de reduzir a complexidade e aumentar a sua escalabilidade. Esta nova arquitectura é denominada *Software-Defined Network (SDN)*.

A Universidade de *Standford* e a Universidade de *California* juntaram-se para o desenvolvimento de um projecto de investigação que tinha como objectivo encontrar uma solução para os problemas e limitações apresentados pela rede actual. O projecto reuniu fabricantes e investigadores de diversas partes do mundo, que sob o conceito de open source, propuseram um novo paradigma e arquitectura de rede, as *SDNs*, que permitem um novo método de definir e transportar o fluxo de informação na rede independentemente do *hardware* utilizado. Sob este pressuposto, as redes deixaram de apresentar um cariz proprietário e passaram a ser redes abertas.

A nova arquitectura propõe a separação do plano de dados e do plano de controlo. Esta separação permite o desenvolvimento de uma entidade lógica central virtual responsável pela configuração e gestão dos elementos de rede que tem uma visão completa da rede. Assim, o gestor de rede, para efectuar a configuração e manutenção da rede, só precisa de interagir com a entidade lógica central, permitindo-lhe abstrair-se dos elementos de rede. Desta forma a integração de novas soluções, aplicações e serviços na rede fica facilitada, sendo necessário menos protocolos para a sua implementação na rede [39] [58].

1.2 Proposta

As *SDNs* são uma nova arquitectura emergente que apresenta uma nova visão para o desenho, desenvolvimento e exploração das infra-estruturas de rede, que baseiam o seu funcionamento na separação do plano de controlo e o encaminhamento dos pacotes, tendo assim maior flexibilidade, abstracção e visão global da rede, com o objectivo de ter um suporte universal, isto é, independente do *hardware*.

Até agora, as *SDNs* têm sido exploradas na sua componente teórica, mas a nível prático da sua implementação poucos resultados têm sido conseguidos. Por isso é necessário o investimento e desenvolvimento de plataformas que permitam verificar os princípios de funcionamento das *SDNs*, como por exemplo, a configuração, administração protecção e optimização dos recursos de redes. Outro ponto importante de estudo é como irá ser efectuada a transição entre as *SDNs* e a arquitectura actual, isto é, como irá ser feita a mudança de equipamentos e redução de custos. Finalmente e não menos importante, encontra-se a segurança e a fiabilidade dos dados que percorrem a rede, já que a centralização da camada de controlo num único elemento de rede pode constituir uma vulnerabilidade, visto que só apresenta um ponto de acesso. É neste seguimento que surge o trabalho proposto por esta Dissertação que pretende desenvolver uma *testbed* baseada em *SDNs* que tem como objectivo testar as funcionalidades e capacidades desta nova arquitectura de rede.

A realização deste trabalho pretende assim a combinação de todos estes pressupostos e conceitos adjacentes para o desenvolvimento de uma plataforma de controlo automático de redes *Openflow*. Entre as suas funcionalidades apresenta o mapeamento de redes, activação de serviços e reoptimização da rede. Neste sentido foram desenvolvidos 4 módulos para dar cobertura à plataforma. O primeiro destes dois módulos é o responsável pela monitorização da rede, isto é, verificar o seu estado e notificar possíveis problemas da mesma. O segundo módulo será o responsável pela activação dos fluxos correspondentes aos serviços que se pretendem activar na rede. Relativamente aos outros dois módulos podemos referir que correspondem a módulos externos cujo objectivo de integração no demonstrador é permitir testar as funcionalidades do demonstrador *Openflow*. Finalmente foram desenvolvidos mecanismos que garantem a comunicação entre os módulos. Todavia tentar-se-á efectuar uma comparação a nível do rendimento apresentado pelas redes utilizadas actualmente e as *SDNs*, a fim de estimar o impacto que as mesmas terão no futuro das redes informáticas.

Este trabalho dará origem a um artigo de conferência internacional que se encontra em preparação e será submetido no verão de 2013.

1.3 Organização da tese

A Dissertação tem um total de seis capítulos que estarão organizados da seguinte forma. O capítulo dois descreve o novo paradigma das redes definidas por *software* e as ferramentas associadas ao mesmo, com o fim de conhecer o princípio de funcionamento, as especificações e limitações que apresentam, para as considerar no projecto da solução a apresentar. De seguida, estudaremos as soluções presentes no mercado para a aplicação e desenvolvimento de uma arquitectura *SDN*, com o intuito de conhecer as opções disponíveis e escolher a que mais se adeque ao nosso projecto. O capítulo três expõe a contribuição proposta para o problema da arquitectura actual de rede, com uma descrição da arquitectura, protocolo e ferramentas a serem utilizadas na solução. O capítulo quatro apresenta a *testbed* desenvolvida para o estudo e análise da solução proposta, assim como o conjunto de funções desenvolvidas para o correcto funcionamento do projecto. O capítulo cinco descreve o ambiente de teste desenvolvido a fim de avaliar o desempenho e funcionamento da implementação da solução proposta, e também inclui algumas conclusões das experiências desenvolvidas. Finalmente o capítulo seis reflecte as conclusões conseguidas ao longo do trabalho e apresenta um conjunto de trabalhos futuros que podem ser realizados com o objectivo de aumentar as funcionalidades da solução proposta.

Capítulo 2

Estado da Arte

2.1 Introdução

Este capítulo serve de enquadramento e ponto de referência sobre todos os tópicos que se encontram relacionados com a temática principal deste documento. A secção 2.2 abordará as *SDNs* dando uma pequena descrição da sua história, os seus pressupostos, a sua arquitectura e a importância na evolução das redes informáticas.

As secções 2.3 e 2.4 incluem uma análise comparativa das *SDNs* com a tecnologia precedente a ela e que permitiu o seu desenvolvimento, isto é, a virtualização e o *CC*.

A secção 2.6 apresentará o leque de tecnologias adjacentes ao mundo das *SDNs*, mais concretamente o *Openflow*.

Finalmente a secção 2.6.3 efectuará um estudo de mercado sobre os diversos controladores e comutadores, e a secção 2.7 apresentará o *software* existente que pode ser utilizado para o desenvolvimento e criação de uma rede *Openflow*.

2.2 *Software Defined Networks (SDN)*

A arquitectura actual da rede foi concebida para que o controlo, encaminhamento e sinalização da rede constituissem uma camada de gestão estática, hierárquica e dependente da infra-estrutura da rede. Esta concepção de projecto desencadeou uma série de problemas e limitações na arquitectura actual da rede, que foram apresentados na secção 1.1.1. Com o objectivo de tentar colmatar estes problemas surgiram as *SDNs*.

As *SDNs* correspondem a uma tecnologia emergente que propõe uma mudança da arquitectura e princípio de funcionamento da rede actual. Mediante a segmentação dos planos de controlo e de dados, torna-se possível olhar para a rede como um todo, já que o plano de controlo é colocado numa camada superior, e não como um conjunto de elementos de redes individualizados [41].

As *SDNs* substituem o plano de controlo dos *switches* e *routers* presentes na rede por uma camada de *software* a fim de tornar a rede mais programável, como por exemplo, proporcionar uma interface de comunicação entre a camada de controlo e a camada de dados [62].

As *SDNs* propõem uma evolução na arquitectura de rede semelhante à verificada na arquitectura de computadores, onde de uma infra-estrutura centralizada e proprietária passou-se para uma infra-estrutura que pode apresentar um controlo distribuído e de cariz *open source* [38].

No início, os computadores foram concebidos sobre uma arquitectura fechada e indivisível, onde todas as suas componentes interligavam entre si num único protótipo que variava de fabricante para fabricante. A evolução tecnológica permitiu simplificar a arquitectura dos computadores para um modelo multi-camadas, constituído por três camadas como a figura 2.1 apresenta, permitindo computadores de interfaces abertas, isto é, que possuam o processador de um fabricante e o sistema operativo de outro favorecendo assim o aumento da sua evolução e desempenho. À semelhança da mudança na arquitectura, encontramos as *SDNs* que propõem um modelo de camadas para suporte, gestão e manuntenção da rede, desacoplando os planos de controlo e dados em duas fatias diferentes, que na actual arquitectura se encontram localizados na mesma área, como ilustra a figura 2.1. Este modelo permite reduzir a complexidade presente nos dispositivos de rede ao concentrar, desde o ponto de vista lógico, a inteligência da rede numa camada. Esta camada é suportada por controladores de *software* e é responsável por gerir as políticas de encaminhamento. Desta forma os elementos de rede deixam de ter que processar um conjunto de protocolos para realizar o encaminhamento de pacotes da rede [38].

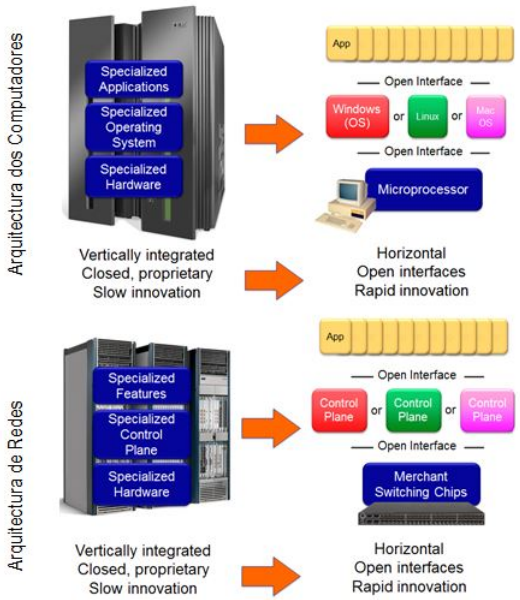

Figura 2.1: Comparação entre a arquitectura dos computadores e a arquitectura das SDN [38]

De acordo com o seu método de funcionamento e processamento dos pacotes, a rede não depende da programação estática de cada um dos elementos da rede, como na rede actual. Verifica-se uma troca de mensagens entre o *software* de controlo (Controlador) e cada um dos elementos de rede tornando assim a rede dinâmica [5] [19].

Estes princípios proporcionam às *SDNs* um conjunto de características desejadas no modelo actual da rede, como por exemplo, o dinamismo que é conseguido pelo controlo sobre todos e cada um dos elementos da rede proporcionado pela entidade lógica virtual, e a flexibilidade dada pela centralização da inteligência da rede numa entidade que permite implementar de forma mais simples e rápida novas aplicações e serviços na rede [28] [19].

2.2.1 Arquitectura

A arquitectura das *SDNs* controla a rede a partir de um dispositivo logicamente centralizado recorrendo a uma infra-estrutura que apresenta três níveis de abstracção como podemos observar na figura 2.2. A primeira camada de arquitectura das *SDNs* apresenta a informação dos diferentes dispositivos físicos e eventualmente de dispositivos virtuais presentes na rede, isto é, *switches* e routers, denominando-se de *Infraestrutucre Layer*. A sua *Application Programming Interface (API)* permite a comunicação com a camada imediatamente superior para configuração da rede.

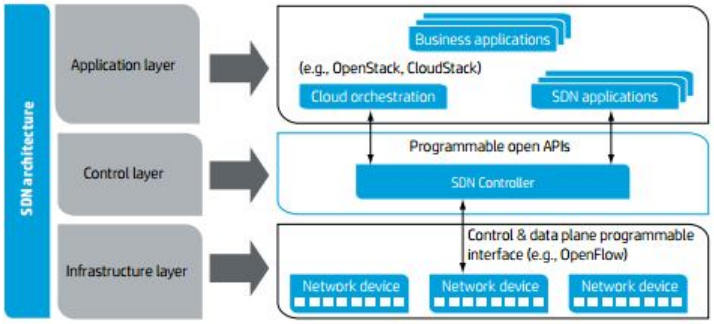

Figura 2.2: Arquitectura SDN [19]

Segue-se a camada de Controlo que é a responsável por configurar, gerir e mapear a rede física aos serviços solicitados pela camada de aplicação. Esta camada encontra-se desvinculada da camada de infra-estrutura, proporcionando assim uma visão geral e centralizada da rede como um todo e não como elementos independentes interligados.

Finalmente encontra-se a camada de aplicação que é responsável por estabelecer o elo de ligação entre os processos aplicativos e os utilizadores que pretendem usufruir de um determinado serviço. Entre as suas funcionalidades podemos destacar o fornecimento de diversas qualidades de serviço a todos os elementos da rede.

Os componentes que fazem parte da arquitectura das *SDNs* são os *switches*, o controlador e as interfaces de comunicação.

Um *switch SDN* é definido como o dispositivo físico (*hardware*) responsável pelo encaminhamento da informação presente nos pacotes de configuração, de controlo, e informação dos algoritmos enviados pelo controlador [41].

O controlador é a aplicação responsável pela gestão do controlo de fluxos para permitir a criação de redes inteligentes. Corresponde ao núcleo de uma rede *SDN* localizando-se na camada de controlo, ou seja, encontra-se entre a camada de aplicação e a de dados, o que lhe proporciona as seguintes funcionalidades: estabelecer as comunicações provenientes da camada de aplicação, aplicá-las na camada de dados e facilitar a gestão da rede [19].

As interfaces de comunicação correspondem ao protocolo ou meio de comunicação utilizado para a troca de informação entre a camada de dados e de controlo. O protocolo normalizado mais utilizado pela indústria e investigação é o *Openflow*. A sua descrição e especificação são desenvolvidas na secção 2.6.

2.2.2 Redes Actuais vs *SDN*

Após apresentar as *SDNs* e a sua arquitectura abordaremos as diferenças entre este novo paradigma e a rede actual.

A fim de reduzir a complexidade das redes informáticas organizou-se a rede numa pilha de camadas de níveis, dando origem ao modelo *Open Systems Interconnection (OSI)* [14].

O modelo *OSI* é uma plataforma de desenvolvimento de protocolos para normalizar a interligação de comunicação entre sistemas, independente da arquitectura específica de cada um deles. Estabelece as linhas a seguir para que o *software* e os dispositivos de redes de diferentes fabricantes funcionem juntos [34]. Por esse motivo, cada camada do modelo *OSI* está encarregue de resolver uma parte do problema, ou seja, cada camada executa funções específicas e independentes das camadas adjacentes.

Cada nível da arquitectura comunica com a camada imediatamente superior ou inferior, propondo serviços à camada superior e utilizando os serviços da camada inferior. Finalmente, cada camada é responsável por encapsular os dados provenientes da camada superior, adicionando as suas próprias informações no pacote que enviará para a camada inferior ou viceversa.

A comunicação entre camadas é feita recorrendo a directrizes genéricas (protocolos). O protocolo é o responsável por definir o formato e a ordem das mensagens trocadas entre as duas entidades ou mais, assim como as acções realizadas na transmissão e recepção da informação.

Os protocolos de rede mais conhecidos são o protocolo *TCP* (Transfer Control Protocol) presente na camada de transporte, o protocolo *Internet Protocol (IP)* na camada de rede. Por outro lado, na camada de aplicação encontramos protocolos de aplicação tais como o *Simple Mail Transfer Protocol (SMTP)* e o *Hypertext Transfer Protocol (HTTP)* que permitem a correcta comunicação entre a rede de dispositivos e a aplicação.

Parte do problema das redes actuais deve-se a que os dispositivos de rede que a constituem (*routers*) apresentam na sua arquitectura interna o plano de controlo, responsável pela tomada de decisões e o plano de dados responsável pelo encaminhamento da informação. Assim, o plano de controlo é executado no próprio equipamento por meio dos protocolos inseridos no seu sistema operativo, não sendo possível trocar qualquer tomada de decisão que não tenha sido prevista nos protocolos, tornando a rede comercial fechada e dependente das soluções do fabricante do equipamento [42][12].

Num contexto paralelo encontramos as *SDNs* que propõem uma arquitectura que desacopla o plano de controlo do plano de dados, permitindo ao utilizador a escolha da sua solução em cada um dos planos, como se pode ver na figura 2.3. Esta metodologia das *SDNs* tem grande impacto a nível económico e computacional, pois não só permite reduzir os custos na compra de dispositivos de rede como oferece a capacidade de testar novas soluções e tecnologias de forma mais rápida e simples. Por outro lado, permite ver a rede como um todo e não como dispositivos individuais, pois a inteligência da rede é concentrada no controlador responsável pela configuração e gestão dos equipamentos [62].

As *SDNs* apresentam uma pilha protocolar de menor tamanho constituída basicamente pela camada de dados, controlo e a camada de monitorização. A abstracção oferecida entre as diversas camadas permite reduzir a carga computacional presente nos elementos de rede, centralizando-a no plano de controlo já que é o responsável pela gestão da rede. Nas *SDNs* basta um único protocolo para garantir a comunicação e troca de informação [42][13].

Outro dos problemas que apresenta a rede actual é a qualidade de serviço, já que seguindo o princípio de *best-effort*, a utilização dos recursos é precária prejudicando claramente a qualidade de serviço oferecida com o aumento de pedidos de activação de serviços. As *SDNs*, pela sua arquitectura e abstracção entre camadas, permitem uma utilização mais eficiente dos recursos, podendo-se activar um número maior de serviços que

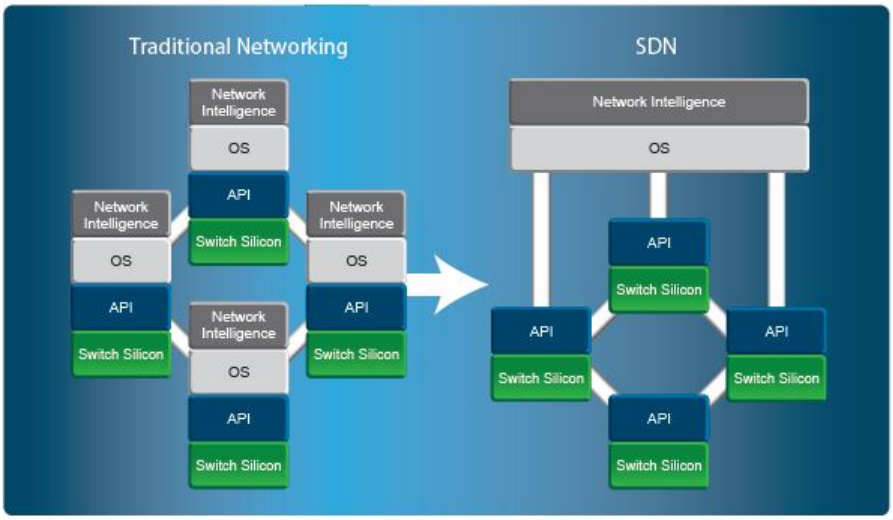

Figura 2.3: Arquitectura Actual vs Arquitectura SDN [46]

---

contribuam para a adesão de um número maior de utilizadores para os mesmos [28].

Finalmente, outra das características que diferencia as redes actuais das *SDNs* são os padrões de tráfego, pois enquanto a infra-estrutura actual foi desenvolvida numa arquitectura cliente-servidor, onde a maior parte do tráfego resultante da comunicação era feita no núcleo da rede, as *SDNs* propõem a mudança dos padrões de tráfego, permitindo que a comunicação seja feita nas extremidades da rede seguindo uma arquitectura servidor-servidor, o que permite reduzir a complexidade inserida na rede para efectuar a comunicação e aumentar a escalabilidade da rede.

2.2.3 Vantagens e Desvantagens das *SDNs*

Entre as vantagens das *SDNs* encontramos:

- Agnóstica ao *hardware*: a deslocação da inteligência da rede para uma camada superior permite que exista uma abstracção da camada inferior, ou seja, os elementos individuais presentes na rede [19].
- Redução da complexidade: a deslocação do plano de controlo para uma camada superior permite que os elementos presentes na infra-estrutura de rede passem a ser elementos muito mais simples que apenas precisem de garantir serviços de conectividade, isto é, a inteligência que hoje está nos elementos passa a estar no controlador [19].
- Facilidade e Inovação: as *SDNs* permitem que haja uma segmentação das redes em camadas. Com esta segmentação é possível ter sob o substracto diferentes redes independentes, permitindo assim o teste de tecnologias disruptivas sem prejudicar o normal funcionamento das outras redes [19].
- Dinamismo, Robustez e Segurança: a actual arquitectura das *SDNs* que tem uma visão e controlo da totalidade da rede permite reagir em tempo real a alterações e problemas.
- A separação do plano de dados do plano de controlo torna a rede mais robusta e segura, já que torna fácil a localização da fonte de problema ou o alvo de ataques, pois o controlador tem a visão geral da rede [19].

As *SDNs* como tecnologia emergente apresentam grande parte das suas desvantagens e problemas em fase de investigação e de teste a fim de as colmatar. Entre as desvantagens das *SDNs* encontramos:

- Imaturidade: a maior desvantagem que as *SDNs* apresentam ao dia de hoje prende-se com a falta de especificações normalizadas para interagir com uma plataforma de gestão de redes existentes, e solucionar os problemas associados à falta de informação.
- Compatibilidade: embora as *SDNs* proporcionem flexibilidade e dinamismo à rede, a integração de determinados serviços e recursos suportados pela rede actual, como por exemplo, as políticas actuais de segurança na sua arquitectura constituiem um problema, sendo necessário estudar a viabilidade de uma solução geral.
- Segurança: Ao centralizar o controlo da rede numa entidade lógica virtual (Controlador) ganhamos dinamismo e flexibilidade, contudo a rede torna-se vulnerável, pois um ataque a esta entidade ou uma falha técnica da mesma põem em causa o desempenho e a fiabilidade da informação transportada pela rede. Por isso é importante o desenvolvimento de uma política unificada de segurança.
- Disponibilidade: outro dos problemas a enfrentar pelas *SDNs* é a disponibilidade e desempenho do serviço prestado pela rede, pois se houver uma falha ou congestionamento na camada responsável pelo controlo de rede, a comunicação entre dispositivos perde-se, podendo causar transtornos no correcto funcionamento da rede.

As *SDNs*, como qualquer outro paradigma encontrado na comunidade científica, apresentam vantagens e desvantagens como podemos observar, sendo necessário realizar uma avaliação ponderada destas para a solução a implementar.

2.3 Virtualização de rede

A virtualização permite que sob um dispositivo físico coexistam vários elementos lógicos independentes que oferecem as mesmas funcionalidades e características que os correspondentes elementos físicos, idealmente sem perdas de desempenho.

A virtualização de rede pode ser definida como um grupo de recursos virtuais (*switches* e *routers*) interligados através de ligações virtuais dedicadas independentemente do *hardware* subjacente, que permite a coexistência de múltiplas redes virtuais no mesmo substrato físico [45]. Por outro lado, a virtualização de rede tem como objectivo a redução dos custos de utilização e desenvolvimento das redes informáticas, e flexibilidade de funcionamento dos protocolos existentes na rede independentemente da infra-estrutura [59] [25].

Os princípios da virtualização são:

- Flexibilidade e heterogeneidade: a flexibilidade nos ambientes virtualizados está associada à programação. Assim o gestor de rede deve ser capaz de programar as funcionalidades de cada um dos elementos de rede e os protocolos adjacentes, assim como todos os tipos de tecnologias da rede.
- Escalabilidade: uma das razões para a utilização da virtualização nas redes prende-se com a coexistência de múltiplas redes no mesmo substrato. Isto implica que a escalabilidade seja um ponto fundamental para a virtualização de rede. O provedor de infra-estrutura não deve restringir a criação de redes virtuais enquanto houver garantias de independência entre redes em termos de rendimento.
- Abstracção e segurança: A abstracção entre dispositivos é importante a fim de evitar que o comportamento anómalo de uma rede virtual possa afectar o funcionamento da restante infra-estrutura, e assim tornar-se um ponto de quebra de segurança que comprometa o desempenho da rede.

- Aumento da eficiência e desempenho da rede: é uma técnica que aumenta o número de recursos existentes na rede a partir de um número pequeno de dispositivos de rede [45].

Os avanços tecnológicos desenvolvidos no âmbito da virtualização permitiram o aparecimento das *Network Functions Virtualization (NFV)* . As *NFV* correspondem a um conjunto de funcionalidades e aplicações de rede, realizadas em *software*, para controlo e gestão da rede. Estas podem migrar facilmente de um ponto para outro da rede, de acordo os requisitos da rede, sem necessidade de instalar as mesmas no novo equipamento ou área de aplicação [11] [60].

Desta forma, pretende-se aumentar a flexibilidade e a elasticidade da rede, assim como facilitar a integração de novas aplicações e serviços na rede. As *NFV* apresentam também a possibilidade de correr num único nó da rede, permitindo assim a redução de custos [4] [52].

2.3.1 *SDNs* e Virtualização de rede

As *SDNs* e a virtualização de rede são dois conceitos independentes que visam dar resposta aos requisitos actuais da rede a fim de solucionar o problema da arquitectura actual. Contudo, o surgimento das *NFV* proporcionou uma nova perspectiva sobre estes dois paradigmas que mostra uma certa complementaridade positiva da co-existência das duas num mesmo ambiente, como podemos observar na figura 2.4.

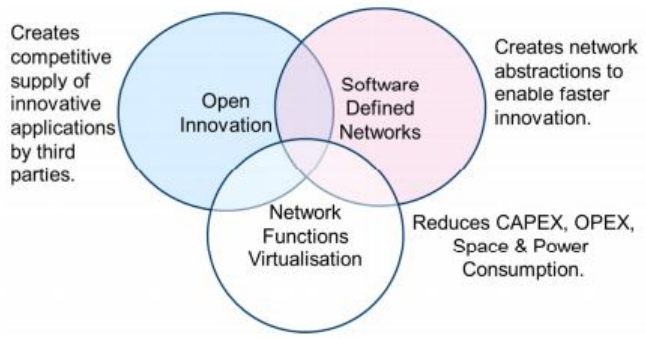

Figura 2.4: Compatibilidade entre SDN e NFV [4]

Por um lado temos as *SDNs* que visam permitir o desenvolvimento e fácil integração de novas tecnologias na rede, assim como melhorar a gestão e manuntenção da mesma. Por outro lado temos as *NFV*que visam reduzir os custos económicos e o espaço ocupado pelos actuais elementos de rede [25] [4].

A separação dos planos de controlo e dados das *SDNs* complementam as funcionalidades das *NFV*, permitindo melhorar o desempenho da rede, simplificar a compatibilidade das implementações existente e facilitar o processo de funcionamento e manuntenção da rede. Por outro lado, as *NFV* proporcionam às *SDNs* uma infra-estrutura de acordo com a arquitectura das *SDNs* permitindo a sua fácil aplicação, já que ao virtualizar as funções e dispositivos de rede, o seu funcionamento fica dependente do plano de controlo. Também as características das *NFV* permitem às *SDNs* executar as funções de rede independentemente do *hardware* dos

elementos de rede. A virtualização de rede com tecnologias *SDNs* e *NFV* permitirá que as funções de redes virtuais sejam facilmente aplicadas a diferentes plataformas de *hardware* [4] [11].

Assim, a combinação das *SDNs* e das *NFV* reflectir-se-á numa redução dos custos económicos, já que não será necessário adquirir novos equipamentos, e no consumo energético das empresas. Com estas tecnologias será possível aumentar a velocidade do tempo de mercado para a integração de uma nova tecnologia. A disponibilidade de dispositivos de rede *multi-tenant* permitirão a utilização de uma plataforma única para diferentes aplicações e utilizadores. Isto permite aos operadores de rede partilhar recursos através de diferentes bases dos clientes. A introdução de serviços específicos baseados em conjuntos geográficos ou de clientes permite que os serviços possam ser escalados mais rapidamente [60] [4][52].

2.4 *Cloud Computing (CC)*

Uma das áreas que tem visto associado o seu nome às redes definidas por *software* é o *CC* e a sua aplicação nos *DataCenter (DC)* .

O *CC* é outro paradigma emergente que tem mudado a forma de armazenamento dos conteúdos de informação por parte dos utilizadores da rede. De acordo com Institute of Electrical and Electronics Engineers (IEEE) é definido como o novo paradigma de Internet para o armazenamento da informação nos servidores de Internet [1]. Para o National Institute of Standards and Technology (NIST) é um modelo para acesso ubíquo de uma forma a-pedido a um conjunto de recursos (computação, armazenamento, rede) que podem ser rapidamente aprovisionados e libertados com um baixo esforço de gestão ou interacção com o provedor de serviços [47].

A definição do NIST enumera cinco características fundamentais para o *CC*: *on-demand self-service*, acesso a rede de banda larga, conjunto de recursos, elasticidade e *pay-per-use*. Todavia faz referência a três modelos de serviço *software*, plataforma e infra-estrutura.

Entre as vantagens do *CC* podemos referir: a utilização de serviços e aplicações sem necessidade de os instalar no dispositivo; ser agnóstico ao *hardware*; permitir o acesso a informação desde qualquer lugar do mundo, bastando que haja acesso à Internet; desta maneira permite diminuir a infra-estrutura física de redes locais, que se traduz num menor consumo de energia que provoca uma redução de custos [47] [32] [7].

2.4.1 *CC* e *SDNs*

Todas estas vantagens favoreceram o aparecimento de unidades de armazenamento que concentram os recursos necessários para o processamento de informação e organização da rede *(DC)*. Contudo a sua gestão e manutenção tem-se tornado um problema para os actuais gestores de rede devido à arquitectura actual de rede. Vejamos algumas das suas características e os problemas que eles apresentam.

Os *DCs* actualmente apresentam milhares de dispositivos físicos, servidores virtuais, aplicações comerciais que, dada a actual arquitectura de rede, faz com que o armazenamento dos dados não seja contínuo, produzindo lacunas na memória que guarda a informação. Esta situação complica o tratamento de dados próximos destas zonas penalizando o desempenho e qualidade do serviço. Por outro lado, os *DCs* apresentam um padrão de tráfego horizontal, pois a troca de informação é feita maioritariamente entre servidores, dada a quantidade elevada de recursos a gerir, o que faz com que os servidores passem a ser dos elementos mais dinâmicos presentes na rede. Este facto obriga-os a ter um conhecimento em tempo real da disposição dos elementos e recursos da rede, perdendo-se escalabilidade e velocidade. Outro dos problemas que os *DCs* apresentam é a segmentação e segurança dos dados que é segurada pela criação de*Virtual Local Area Network (VLAN)* , sub-redes IP e *firewalls* a fim de garantir a fiabilidade dos dados armazenados [62] [41].

O novo modelo de rede *SDNs* em conjunto com o protocolo *Openflow* permitem, dada as suas características, uma gestão dos *DCs* mais fácil devido à centralização do plano de controlo numa camada, que permite ao gestor de rede uma visão geral da rede. Outra das vantagens adjacentes a esta nova arquitectura é que permite a centralização das tabelas de fluxos para a programação e gestão do tráfego, o que permitiria a eliminação das lacunas. Finalmente este princípio das *SDNs* permite simplificar a arquitectura de segurança de rede, orientando-a aos fluxos de serviços controlada por uma *firewall* [16] [55].

Estas características das *SDNs* oferecem um conjunto de vantagens e mais-valias para o *CC*, tais como uma maior flexibilidade e escalabilidade na gestão e manuntenção dos *DCs* advindas da separação entre o plano de dados e controlo. Por outro lado, torna mais rápido a troca de informação entre dispositivos, visto que quer os *DCs* quer as *SDNs* apresentam um fluxo de tráfego horizontal. Estas características em conjunto com os princípios do *CC* permitem uma redução de custos, pois são precisos menos dispositivos de redes para efectuar o direcionamento do tráfego, assim como se reduz a carga computacional da rede [32] [7].

A prova de que as *SDNs* são a arquitectura actual melhor preparada para as necessidades da *CC* foi a experiência realizada pela Google na sua rede local de *DC* em 2012, que permitiu interligar os seus *DCs* e gerir de modo eficiente as diversas variedades de tráfego que circulam na rede, o que se traduziu na redução dos custos da empresa num total de 8%. O sucesso dado por esta experiência levou a google a lançar para 2014 a primeira rede local desenvolvida num ambiente *SDNs* enquadrada no âmbito do projecto *G-Scale* que pretende o estabelecimento de comunicação entre *datacenters* [16].

Em conclusão, as *SDNs* apresentam requisitos dinâmicos e implementações flexíveis que podem contribuir para superar as dificuldades que hoje apresentam os *datacenters* na actual arquitectura de rede.

2.5 Protocolos e Tecnologias

Entre os protocolos que dão suporte às *SDNs*, o mais normalizado e aceite é o *Openflow*. Contudo, a ideia de redes geridas por *software* teve origem uns anos atrás, existindo protocolos e tecnologias antecessoras do *Openflow*, como por exemplo, o Open Signaling (OPENSIG) , Network Configuration Protocol (NETCONF) e ETHANE [62] [41].

O *Open Signaling* [8] é a primeira tecnologia que surge com um princípio de funcionamento semelhante ao das *SDNs*, tendo chegado a desenvolver redes de caixas automáticas, Internet e telefónicas sob este princípio. Eles propunham o desenvolvimento de interfaces abertas e programáveis que permitissem a integração e desenvolvimento de novos serviços na rede. Motivados por estas ideias e pressupostos chegaram a desenvolver o protocolo *Global Standards Management Process (GSMP)* [41] que, implementado num *switch*, permitia estabelecer e desligar ligações de rede, gerir os portos do *switch*, solicitar a informação de configuração e adicionar ou remover novos dispositivos num grupo de uma ligação multicast. Contudo, dado o cariz propietário e fechado da arquitectura de rede, houve um conjunto de limitações e problemas que tornava quase impossível a integração de novos serviços e aplicaçoes de rede.

O *NETCONF* [40] corresponde a um protocolo de configuração de dispositivos de rede dentro da área de operações do *Internet Engineering Task Force (IETF)* . Este protocolo pretende unificar o processo de configuração dos dispositivos de rede e colmatar as limitações e problemas apresentados pelos protocolos da arquitectura actual, como por exemplo o *Simple Network Management Protocol (SNMP)* . O *NETCONF* apresenta um total de 6 instruções simples que permitem criar, editar, copiar e remover configurações, e ainda obter informação estatística e do estado dos elementos de rede. Contudo, o *NETCONF* não é totalmente programável, pois qualquer nova funcionalidade teria que ser implementada quer no dispositivo de rede quer no gestor de rede. Por isso podemos concluir que é um mecanismo que surgiu com o intuito de tornar a

configuração dos dispositivos de forma automatizada.

O antecessor do *Openflow* foi o projecto *ETHANE* [9] que propôs à comunidade científica uma nova arquitectura de rede empresarial, gerida por um controlador centralizado responsável pela gestão e manutenção da política de encaminhamento e segurança na rede. O seu método de funcionamento é realizado pelo controlo de acesso. O controlador também é responsável pelo envio ou descarte de um pacote na rede [41].

O *Openflow* surgiu em 2007 na Universidade de *Stanford* [39]. O aparecimento do *Openflow* permitiu experimentar protocolos, de *software* aberto, para diferenciação de tráfego e configuração da rede de uma forma mais eficiente e flexível.

A descoberta do *Openflow* marcou o início de uma nova etapa das redes informáticas que, devido ao seu sucesso, gerou uma nova procura por parte das empresas proprietárias, a fim de oferecer a melhor solução do mercado. Nesta procura pelo novo paradigma que solucione os problemas apresentados pela rede de hoje surgem duas vertentes. A primeira, a vertente do *OpenFlow*, defende que o controlador deve estar num dispositivo independente, com conhecimento total da rede, facilitando assim a gestão da rede, já que passamos a ter acesso a todos os recursos de forma remota. Contudo, a centralização do plano de controlo levanta uma série de questões e considerações, como por exemplo, a localização física do controlador assim como o desenvolvimento de um protocolo para garantir a comunicação interna. Por outro lado temos a segunda vertente que propõe uma distribuição do controlador pelos *switches* presentes na camada física da rede, isto é, distribuir a inteligência e *software* do controlador pelos nós do substrato, apresentando como vantagem o facto de não ter que desenvolver o protocolo interno para comunicação, assim como um aumento do desempenho nas comunicações estabelecidas entre os clientes e a infra-estrutura de rede melhorando o funcionamento das *SDNs*. Entre as tecnologias mais conhecidas temos o *Openstack, NETCONF, VMWare* e outros *hipervisor*.

No capítulo seguinte é introduzido o *OpenFlow* com mais detalhe.

2.6 *Openflow*

Como vimos no estudo da arquitectura das *SDNs*, a solução para a interface de comunicação com maior aceitação no mundo da investigação é o *Openflow*. Nas secções seguintes efectuaremos um enquadramento do mesmo.

2.6.1 *Openflow Protocol*

O *Openflow* é um protocolo de comunicação que proporciona uma interface normalizada para a troca de informação entre o plano de dados e de controlo. Utiliza um *software* de controlo externo, responsável por controlar o encaminhamento da informação de um *switch*. O controlador é o responsável pela gestão das tabelas de encaminhamento e dos fluxos de dados que podem ser implementados na rede.

O plano de dados é constituído por uma tabela de fluxos associada a cada *flow* que entra na rede, comparando os dados do pacote que chega com a informação presente no *switch*, fazendo o descarte ou envio do pacote.

O plano de controlo é constituído pelo controlador que efectua a gestão e manutenção dos fluxos na rede, ou seja, é responsável pela inserção, actualização e eliminação de fluxos para não sobre-carregar o sistema.

A principal abstracção utilizada na especificação *Openflow* é o conceito de fluxo. O fluxo é constituído pela combinação dos campos do cabeçalho do pacote a ser processado pelo dispositivo. Corresponde às *flags* activas que permitem definir como deve ser direccionado o fluxo pelos dispositivos de rede. Também se encontram presentes os padrões de uso, aplicações e recursos utilizados pelo mesmo, determinando a forma de encaminhamento e as acções a efectuar nos pacotes correspondentes a esse tipo de tráfego. Por exemplo,

podemos considerar que um serviço de baixa prioridade segue sempre o caminho mais longo a fim de evitar o congestionamento de dispositivos que estejam a ser utilizados por um serviço com uma prioridade superior.

O princípio de funcionamento do *Openflow* é o seguinte, como ilustra a figura 2.5:

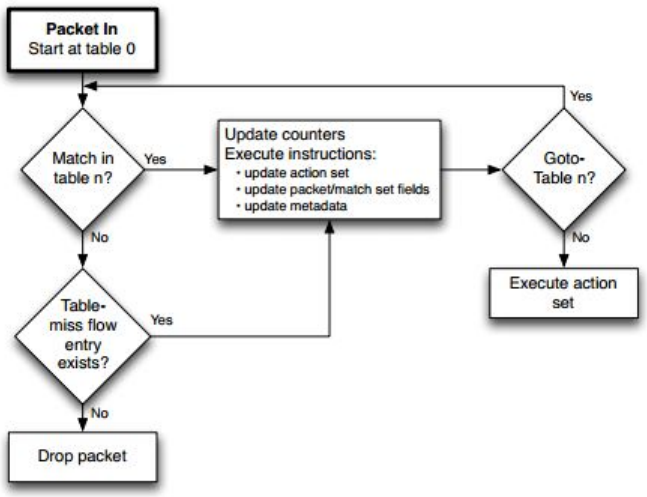

Figura 2.5: Funcionamento do *Openflow Protocol* [3]

- Um *flow* chega a um equipamento *Openflow*, normalmente um *switch* virtual, previamente configurado para o tráfego *Openflow*.
- Seguidamente, o *switch* compara os campos do cabeçalho do pacote com a sua tabela de *flows* introduzidos, a fim de saber se tem alguma configuração para esse tipo de pacote, podendo-se verificar duas situações. No caso de não existir um *flow* previamente para o tipo de pacote, o *switch* envia o pacote para o controlador que toma a decisão sobre a acção a realizar no mesmo. Neste caso, o controlador pode optar pelas seguintes operações: descarte do pacote, sendo lançado um evento com a informação *'miss table'*, que sinaliza ao controlador e ao *switch*, que para esse tipo de pacote não apresenta regra que defina o tráfego; realiza a inserção de um *flow* de acordo com as características do pacote, que permita fazer o *forward* do pacote, isto é, propagar o mesmo. Depois envia-o novamente para o *switch* em que se pretende activar esse tipo de fluxo. No caso de existir a configuração de *flow* para o pacote em questão, e se verificar o *match* com a tabela de *flows*, o pacote é propagado consoante a configuração do tipo de pacote.
- Seguidamente são actualizados os contadores e as acções correspondentes são realizadas.

A especificação 1.3.1 [3] em Junho de 2012 prevê a integração do protocolo *Multi Protocol Label Switching (MPLS)* , a inclusão da tabela *Meter Table* que permite implementar acções de *QoS*, como por exemplo, a largura de banda, utilização de filas de espera configuradas para um determinado porto e também fornece informações estatísticas sobre os diversos fluxos. Também esta nova especificação contempla a utilização dos portos para o tráfego *TCP* e *UDP* à semelhança do que acontece nas redes Ethernet actuais. Todavia, a especificação prevê um conjunto de *testbeds* que permitam analisar o desempenho, monitorizar e gerir a rede. Contudo, a especificação normalizada e a que apresenta um maior número de ferramentas e funcionalidades no

mundo de investigação ao nível da camada de controlo é a especificação 1.0.0 [2]. Os componentes que fazem parte da arquitectura interna do *Openflow* são o *Openflow switch*, o *Openflow controller* e o *Openflow channel*.

2.6.2 *Openflow Switch*

Um *Openflow switch* consiste numa ou mais tabelas de fluxos e uma tabela de grupo. A troca e reenvio de pacotes entre os *switches Openflow* e o controlador é feita pelo *Openflow Channel* e os *Openflow ports* como podemos observar na figura 2.6. Os *switches* comunicam mediante o protocolo *Openflow*, através do qual o controlador adiciona, actualiza e elimina fluxos das tabelas do *Openflow switch* como ilustra a figura 2.6.

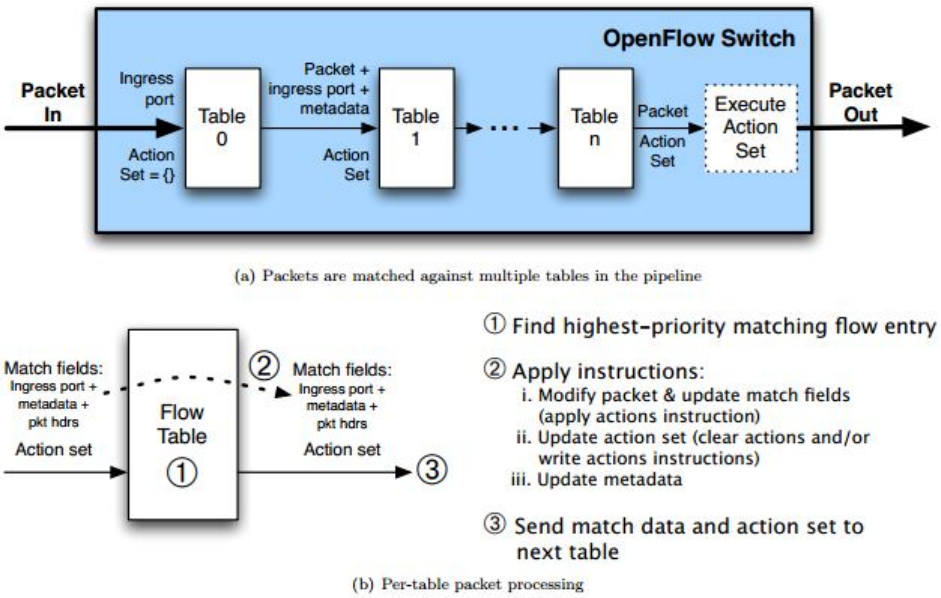

Figure 2: Packet flow through the processing pipeline

Figura 2.6: Funcionamento do *Openflow switch* [3]

Existem dois tipos de *openflow switch*, os *Openflow switch only* que são compatíveis unicamente com o protocolo de comunicação *Openflow*, e os *Openflow hybrid* que, além de permitir o protocolo de comunicação *Openflow*, apresentam suporte para os restantes protocolos presentes na rede actual.

2.6.3 *Openflow Controller*

O *Openflow controller* é parte integrante de um sistema *Openflow* responsável por garantir a comunicação entre os equipamentos intermediários através do protocolo *Openflow*, e realiza operações de configuração, gestão sobre as tabelas de fluxo do mesmo, como apresenta a figura 2.7. Tem como função adicionar e remover entradas na tabela de fluxos em nome da aplicação que está a utilizar o sistema.

Pode-se dizer que o controlador corresponde a um sistema operativo de redes responsável pela gestão e manutenção da rede. Assim sendo, ele é responsável por fornecer a interface de alto nível para as aplicações utilizarem os recursos de *hardware* e também controlar a interacção entre aplicações.

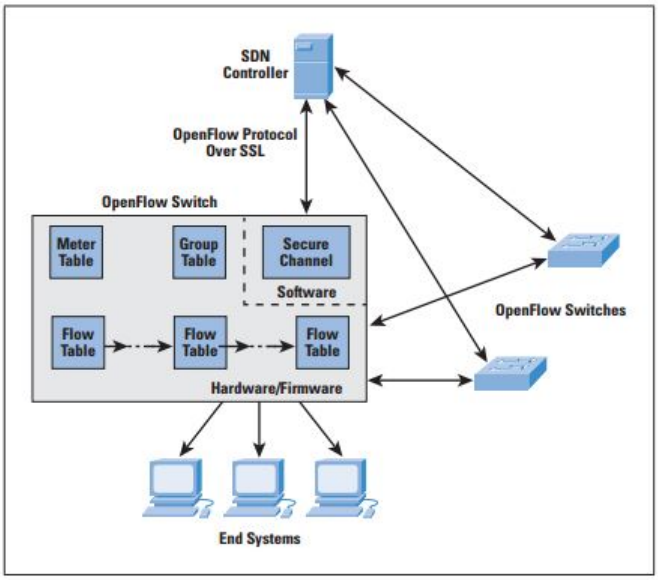

Figura 2.7: SDN Controller

Existem vários tipos de controladores *Openflow*, cuja especificação é fornecer uma camada de abstracção entre a camada de infra-estrutura e a camada de controlo, como se pode observar na figura 2.7, cujas implementações vão desde *Python* até Ruby. Por outro lado, também se verifica o desenvolvimento de outro tipo de controladores com um fim específico, como por exemplo saber o encaminhamento do tráfego *Openflow*, efectuar virtualização entre outros.

Nesta secção pretende-se fazer uma abordagem geral das diversas alternativas que temos para o controlador.

2.6.3.1 *NOX*

A primeira tecnologia *SDNs* criada para obter a informação correspondente à rede e proceder à sua gestão e manutenção foi o Nox [36]. O primeiro controlador *Openflow* foi desenvolvido pela *Nicira*, e posteriormente doado aos investigadores, a fim de chamar a atenção de novos utilizadores e clientes *SDNs* que fornecessem novas ideias para o desenvolvimento de uma solução geral e robusta para as necessidades de todos os utilizadores da actual arquitectura de rede.

Entre as características do controlador Nox, fornece uma visão de rede única armazenada num servidor independente, formada com base em informações como a topologia, localização e serviços oferecidos. O controlo sob a rede é feito ao nível do fluxo, e foi desenvolvido na linguagem de programação C++, embora permita *scripts* e aplicações escritas em *Python*. O seu princípio de funcionamento baseia-se em dois conceitos: o componente e o evento. O primeiro está relacionado com as funcionalidades e características que o *Nox* permite, e o segundo prende-se com a acção a ser aplicada na rede sobre um determinado fluxo [16][56] [22].

Embora o Nox ofereça um ambiente programável, centralizado e com bom desempenho aos investigadores, para redes muito exigentes falha no paralelismo e versatilidade de utilização dos recursos, isto é, apresenta pouca escalabilidade. O Nox não permite executar vários eventos em simultâneo, o que faz com que a sua aplicação e utilização seja feita como tutorial e para dar os primeiros passos em redes *Openflow*. Outra menos-valia é o facto de ser um controlador proprietário [31] [41].

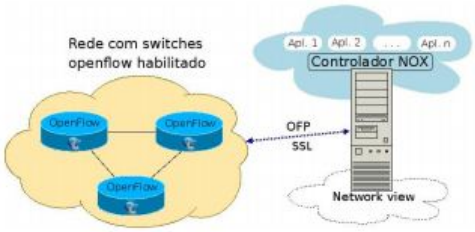

Figura 2.8: Nox [36]

2.6.3.2 *POX*

O *Pox* [36] tem as suas origens no primeiro controlador *Openflow*, o *Nox*, e surge na necessidade do desenvolvimento de uma aplicação *SDNs* para gestão de redes domésticas. A linguagem de programação utilizada é o *Python*. Os seus criadores pretendem que substitua o *Nox* nas aplicações em que o rendimento não seja um requisito crítico.

Em relação ao seu antecessor e aos primeiros controladores *Openflow* que foram orientados para a troca de mensagens *Openflow* recorrendo a primitivas básicas, o *Pox* é organizado através da noção de apresentar uma visão global de rede. Assim as aplicações não precisam de efectuar troca de mensagens, mas poderão monitorizar a rede e alterá-la de acordo ao objectivo pretendido, isto é, os recursos a disponibilizar [42] [41].

O *Pox* também possui um analisador de protocolos que acompanha cada fluxo na rede local e reconstrói o fluxo de aplicação para os pontos identificados como relevantes pelo protocolo para o serviço a activar. A reconstrução da rede continua até que o controlador possua uma noção consistente da rede para avaliar o tráfego consoante as regras inseridas para o encaminhamento do mesmo [16].

2.6.3.3 *SNAC*

O controlador *SNAC* [22] é um controlador de redes corporativas para a configuração de dispositivos com uma linguagem de definição de políticas flexíveis para realizar procuras com a ajuda de padrões de alto nível. Contudo, não fornece um ambiente de programação genérica.

À diferença dos controladores expostos nesta secção, o *SNAC* é um controlador utilizado para monitorizar a rede, mediante a utilização e desenvolvimento de uma interface *web*. O seu código fonte e as suas aplicações baseiam-se no controlador *Nox* anteriormente exposto, e propõem uma interface gráfica para implementar uma política de gestão de rede com controlo centralizado [31].

2.6.3.4 *Beacon*

O *Beacon* [31] é um controlador *Openflow* criado na Universidade de *Standford*, com a linguagem de programação *Java*. Apresenta uma estrutura modular com uma interface *web* que permite visualizar e monitorizar os fluxos, *switches* e topologia presentes na rede. Procura ser dinâmico permitindo alterações e actualizações em tempo real de alguns ficheiros de configuração. Permite eventos como *threads*. O maior inconveniente é que, para o seu desenvolvimento, necessita de ter conhecimento da *Open Services Gateway Initiative (OSGI) framework* [22].

2.6.3.5 Maestro

Maestro [41] é uma plataforma escalável de controlo para *switch Openflow* que fornece interfaces para implementações modulares de aplicações para controlo de rede para aceder, modificar e actualizar a informação sobre a rede, assim como os fluxos presentes na mesma e ainda a manutenção do estado de rede.

Permite a introdução de novas funcionalidades de controlo personalizadas, adicionando módulos de controlo que permitam efectuar uma diferenciação e gestão de rede de acordo com as necessidades de cada utilizador. Possui módulos que criam uma visão global da rede, e módulos para implementar protocolos de *routing*, pois o controlador é o responsável para estabelecer a conectividade inicial entre os diversos dispositivos da rede [56] [62].

A mais valia que o Maestro oferece ao gestor de rede é o paralelismo dentro de uma única máquina para melhorar o desempenho por meio da utilização de *threads*, e a personalização do plano de controlo de acordo com as necessidades do utilizador [31] [16].

2.6.3.6 *FlowVisor*

O *FlowVisor* [61] é um controlador especializado que utiliza o protocolo *Openflow* para controlar a rede física adjacente. Conta com uma camada de abstracção de *hardware* entre as rotas de controlo e as de reenvio de dados que permite comportar-se como um proxy transparente entre *switch* e múltiplos controladores *Openflow*.

A grande diferença que o *flowvisor* apresenta em relação aos outros controladores *Openflow* é a abstracção da arquitectura do controlador que permite criar um conjunto de segmentos de recursos de rede, isto é, dividir a camada de infra-estrutura em sub-redes beneficiárias de um recurso, atribuindo a cada uma delas um controlador diferente, assegurando que cada controlador pode observar e controlar o seu próprio segmento, enquanto que se mantém isolado relativamente ao outro, como se pode observar na figura 2.9. Cada segmento de rede possui uma linguagem de definição de políticas de utilização (descritas por ficheiros isolados de configuração), onde são especificados os limites de utilização dos recursos do segmento, assim como a localização do controlador [61].

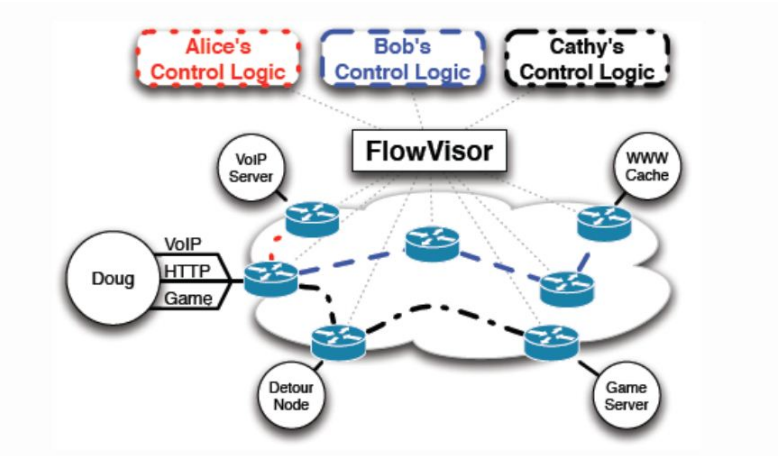

Figura 2.9: FlowVisor [61]

2.6.3.7 *Floodlight*

Floodlight [64] é um controlador *Openflow* para redes corporativas baseado na linguagem *Java*, tendo a sua origem no controlador *Beacon*.

As principais vantagens que o *Floodlight* apresenta relativamente a outros controladores é ter um núcleo de configuração e gestão bastante genérico e flexível, que permite apresentar uma arquitectura orientada ao serviço, como ilustra a figura 2.10, permitindo assim o desenvolvimento de novas aplicações e ferramentas que facilmente são integradas no controlador e consequentemente na rede, de acordo com as necessidades de cada utilizador. Também o facto de ser orientado ao serviço permite que a comunicação entre os módulos ocorra através de serviços, bastando ter que garantir a segurança e fiabilidade dos mesmos [64]. Outra das mais valias que o *Floodlight* apresenta é o facto de ser compatível com redes não *Openflow*, isto é, pode ser utilizado para redes *ethernet*, o que permite realizar um estudo aprofundado sobre as diferenças entre os dois tipos de redes, contribuindo assim para melhorias significativas nas *SDNs*.

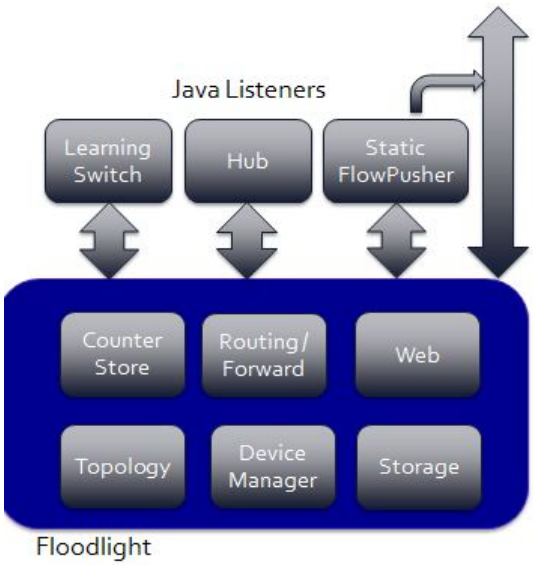

Figura 2.10: floodlight1 [64]

2.6.3.8 *Frenetic*

Frenetic [31] é um sistema baseado numa linguagem funcional desenvolvido para programar redes *Openflow*. O *Frenetic* fornece ao utilizador uma noção completa da rede permitindo a programando a mesma. É composto por duas sub-linguagens integradas: uma declarativa de consultas para classificar e adicionar tráfego à rede, e uma biblioteca reactiva funcional para descrever políticas de envio de pacotes.

O *Frenetic* permite alterar o tráfego da rede em tempo real mediante a adição de regras a baixo nível para os *switches*, mantendo a abstracção da camada superior que não verifica nenhuma alteração no funcionamento da rede e da aplicação, isto é, a aplicação e a topologia da rede não se ressentem da adição do fluxo. Isto é possível porque *Frenetic* trata cada pacote de forma individualizada, isto é, analisa de princípio ao fim, traduzindo-o em comandos *Openflow* que garantem o maior desempenho e flexibilidade da rede. Desta forma diminui-se a quantidade de informação necessária a armazenar na camada física. Por outro lado permite

incluir padrões de tráfego a alto nível para todos os pacotes que estão a passar na rede, recorrendo para tal a parâmetros de agrupação como por exemplo a hora, contadores de pacotes recebidos e enviados, etc.

Actualmente, o *Frenetic* é utilizado como uma aplicação sobre o controlador *Nox*, mas a utilização do mesmo é prescindível.

2.6.3.9 Comparação entre os controladores

Depois de termos apresentado as características de cada um dos seus controladores, resumem-se na tabela 2.1 as principais características deles com o intuito de ressaltar as suas diferenças.

O *Nox* é o controlador *Openflow* de referência, entre as suas características encontra-se o facto de ser um controlador centralizado que precisa de ter conhecimento total do substrato da rede, o qual deve ser previamente armazenado num servidor. O *Pox* é um controlado desenvolvido para redes domésticas, é o sucessor do controlador *Nox*, seguindo assim os mesmos princípios de funcionamento deste. O *Frenetic* é um controlador baseado numa linguagem funcional para programar as redes *Openflow* que permite, pela sua arquitectura interna, analisar cada pacote de forma individualizada diminuindo assim a quantidade de informação necessária a armazenar na camada física. Também permite consultar, classificar e descrever políticas de envio de pacotes.

Por outro lado encontram-se os controladores SNAC, *Beacon* e Maestro que foram desenvolvidos para a monitorização e controlo da rede, permitindo a alteração em tempo real da informação dos fluxos. O SNAC segue um princípio de funcionamento semelhante ao *Nox*, isto é, controlo centralizado. O *Beacon* procura introduzir certo dinamismo ao seu funcionamento mediante o fornecimento de um ambiente *multithread*. Finalmente o Maestro fornece interface para implementações modulares de aplicação para a criação de uma visão global da rede e para implementar protocolos num ambiente *multithread*.

Finalmente tem-se o *FlowVisor* e *Floodlight* que são controladores que fornecem a possibilidade de controlo distribuído. O *FlowVisor* apresenta uma arquitectura interna que permite dividir a camada de infra-estrutura em sub-redes beneficiárias de um recurso atribuído. Assim cada sub-rede possui uma linguagem de definição de políticas de utilização do recurso. O *Floodlight* apresenta uma arquitectura interna orientada ao serviço permitindo assim o desenvolvimento de novas aplicações e ferramentas facilmente integráveis ao controlador e consequentemente na rede.

2.6.4 *Openflow Channel*

O *Openflow Channel* corresponde à interface que interliga o *Openflow switch* ao *Openflow controller*, isto é, a ponte de comunicação entre a camada de dados e a camada de controlo. Através desta interface, o controlador configura e gere a rede *Openflow* (camada *Layer 2 (L2)*), sendo o responsável por receber e gerir as entradas do *switch* até ao controlador e vice-versa.

A interface de acesso recomendada é o protocolo *Secure Sockets Layer (SSL)* . As interfaces alternativas incluem *TCP* e *Packet Capture (PCAP)* e são especialmente úteis em ambientes virtuais.

Um *Openflow controller* pode gerar vários *Openflow channel* para diferentes *switches*, mas também é possível um *switch Openflow* ter um canal diferente para cada controlador *Openflow*. O *switch Openflow* deve ser capaz de estabelecer a comunicação com o controlador previamente configurável pelo utilizador. Depois da ligação estar estabelecida, cada lado da ligação deve enviar uma mensagem *HELLO* com os campos da versão do protocolo *Openflow* a correr, assim como outros dados que permitem a sincronização e ligação entre os equipamentos.

No caso em que um *switch* perde o contacto com todos os controladores como resultado da espera do *echo*, tempo de espera ou de ligações, o controlador deverá entrar imediatamente em modo de segurança ou falha

Comparação entre os controladores			
Nome	**Linguagem**	**Plataforma**	**Característica**
Nox	C++, *Python*	Linux	Controlador de referência *Openflow*.
POX	C++, *Python*	Windows, Linux, Media Access Control (MAC)	Evolução do *Nox* para redes domésticas.
SNAC	C++	Linux	Monitorização de redes *Openflow*
Beacon	*Java*	Windows, Linux, Android	Controlador com ambiente multithreaded.
Maestro	*Java*	Windows,Linux, MAC	Procura aproveitar o paralelismo dos elementos de controlo.
FlowVisor	*Java*	Linux e Windows	Subdivisão da rede por camadas, controlo distribuido.
Floodlight	*Java*	Windows, Mac, Linux	Arquitectura orientada ao serviço, integração de redes não *Openflow*.
Frenetic	Funcional	Linux	Programação da rede como um todo.

Tabela 2.1: Tabela de *Comparação entre os controladores*

de modo autónomo.

2.7 Projectos e Plataformas

As grandes empresas de redes de comunicações juntaram-se para constituir uma organização sem fins lucrativos Open Networking Foundation (ONF) [19] que se encarregará do desenvolvimento e promoção das *SDNs*.

Com esta organização pretende-se o desenvolvimento de novas normas no âmbito das redes de comunicações com o fim de evoluir as *SDNs* para que a combinação das *SDNs* permitam realizar o controlo do tráfego de uma forma mais flexível em qualquer tipo de rede.

A organização pretende utilizar a interface para o controlo de transmissão de pacotes de dados *Openflow*. Os protocolos desenvolvidos serão totalmente abertos a fim de aumentar a velocidade de convergência para uma solução que permita desvincular o fluxo de dados e o fluxo de controlo dos dispositivos interligados.

Outra das organizações que visa dar suporte às *SDNs*, começando pelo *Openflow*, é a *OpenDayLight* [30], apoiada pela *Linux Foundation, Cisco, VmWare, Ericsson, HP, Microsoft e Intel*. Esta organização pretende o desenvolvimento de um conjunto de tecnologias *SDNs*, que incluem um controlador de interfaces, a fim de permitir a integração de aplicações e ferramentas de gestão de *CC* entre outras, como se pode observar na figura 2.11.

A *OpenDayLight* propõe a projecção e desenvolvimento de um ambiente aberto baseado numa arquitectura dinâmica que permite reduzir a complexidade operativa, facilita a abstracção da rede, assim como disponibiliza novos serviços e capacidades que assentem sob o *Openflow*.

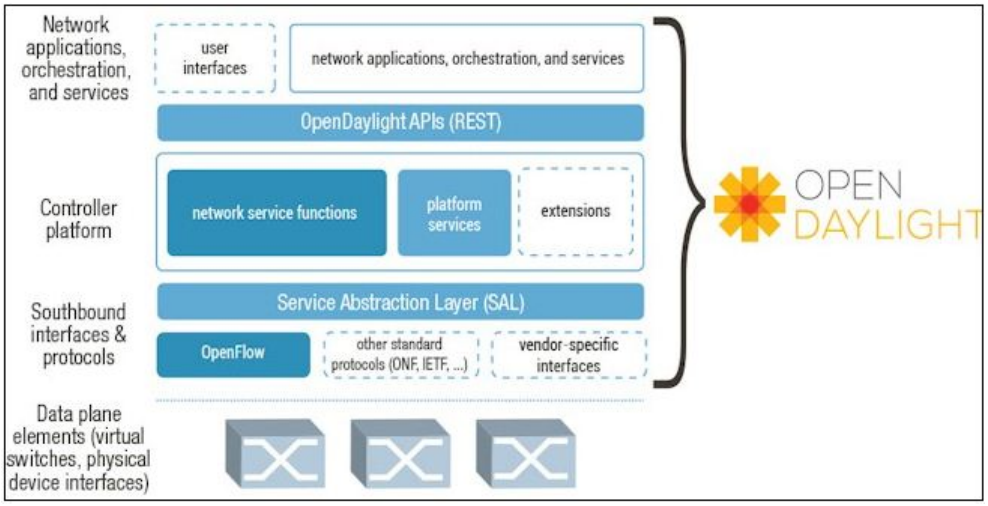

Figura 2.11: Open Day Light [30]

Embora o *Openflow* faça parte da arquitectura do *OpenDayLight*, esta ambiciona ser o mais extensa e flexível possível a nível do controlador, como se pode observar na figura 2.11. Ela pretende oferecer a sua plataforma e código fonte para controladores de cariz proprietário com o intuito de que a diversidade de informação permita o desenvolvimento de um produto de qualidade, robusto e que dê resposta às necessidades do mercado.

De seguida apresentam-se algumas empresas e plataformas.

2.7.1 *Nicira*

A *Nicira* é uma empresa que aposta no *Openflow* e está a desenvolver uma solução proprietária para as redes virtuais, baseando-se na utilização de túneis (*tag VLAN, MAC*, entre outros), para estabelecer uma ligação entre o *hipervisor* e os *switches*.

O sistema é gerido por um sistema de controlo inteligente e distribuído que permite aumentar o número de redes virtuais com o intuito de maximizar o desempenho dos recursos disponíveis na rede, aumentando, em simultâneo, o número de aplicações prestadas.

A sua implementação baseia-se em 3 camadas como se pode observar na figura 2.12:

- *Intelligent Edge:* corresponde ao *OpenVswitch* com a funcionalidade de efectuar o controlo remoto. Localiza-se entre os *hipervisor* e os servidores, constituindo uma camada de abstracção de *software* entre a infra-estrutura física e a rede física.
- Hipervisor: dispositivo físico que interliga a camada *Intelligent Edge* com o *cluster controller*, ou seja, corresponde à ligação entre o campo de dados e o campo de controlo.
- *Cluster Controller:* corresponde a um sistema distribuído de alta disponibilidade que gere todas as componentes da rede virtualizada e as suas ligações.

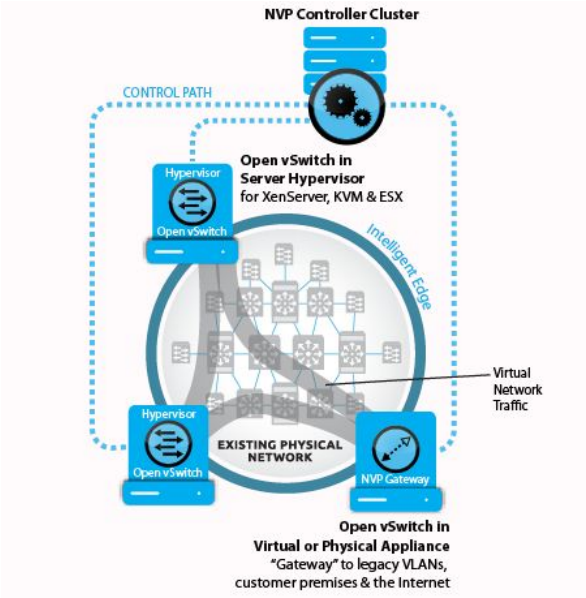

Figura 2.12: Solução *Openflow* da *Nicira* [48]

2.7.2 *HP*

A *HP Networking* foi essencial para o desenvolvimento da tecnologia *Openflow*, desde os seus primeiros passos, já que foi a primeira empresa a desenhar dispositivos comerciais que permitissem a realização de testes de *Openflow*, como se pode observar na figura 2.13, facilitando assim o estudo e o desenvolvimento de um projecto para a criação e constituição de uma norma de uso comercial.

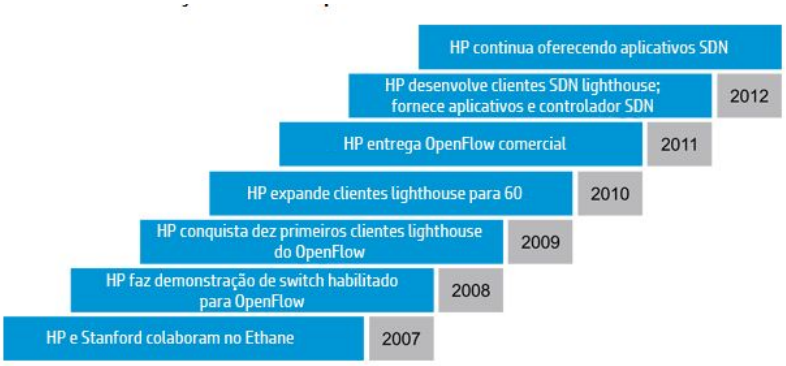

Figura 2.13: HP History [5]

A *HP* também desenvolveu o *Intelligent Management Controller (IMC)* que permite obter o controlo da rede desde um estado único bastando um controlador, a nível lógico, para efectuar a sua gestão. O controlador de redes de aplicativos virtuais oferece uma solução de controlador extensível, escalável e resiliente; utiliza o *Openflow* para programar cada uma das camadas de infra-estrutura. Contudo, a falta de convergência e da especificação de um controlador único tem feito com que a *HP* se questione sobre o desenvolvimento de um

controlador *HP* real, prevendo-se que o lançamento do mesmo seja no segundo semestre de 2013.

Este controlador pode correr em Linux e é capaz de monitorizar e gerir o plano de dados e de controlo de milhões de *switches* e *routers* presentes na rede.

2.7.3 *Cisco*

A *Cisco* propõe uma solução conservadora e diferente da apresentada pelas outras empresas e projectos. Para a *Cisco* a solução não passa por desacoplar o plano de controlo do plano de dados, mas fornecer ferramentas e aplicações que permitam extender a programabilidade da rede para todas as camadas da arquitectura.

A *Cisco* pretende responder aos requisitos das *SDNs* de três formas diferentes. Numa primeira fase fornecerá a possibilidade do desenvolvimento de redes orientadas a protocolos *SDNs* normalizados, como por exemplo, o *Openflow*. Numa segunda fase dará o suporte necessário para redes virtuais tais como o *LISt Processing (LISP)* e o *Virtual Extensible LAN (VXLAN)* [43], com o intuito de juntar o mundo físico e virtual. E finalmente o desenvolvimento de *software* que permita configurar e controlar os *routers* e *switcthes* através de uma *API* universal como podemos observar na figura 2.14 [20].

Neste âmbito a *Cisco* apresenta o controlador *SDN Open Network Environment (ONE)* , desenvolvido em *Java*, que pretende concentrar a inteligência presente na camada de controlo e na camada de infra-estrutura, na camada de serviços de rede e gestão. O controlador permite, mediante a exposição de *APIs*, fornecer ao gestor de rede um conjunto de protocolos normalizados e modelos de implementação de integração de nova informação, que pretendem tornar o *hardware* programável, aumentando a sua escalabilidade e reduzindo a complexidade [43][15].

Embora o *ONEpK* corresponda a uma *API* propietária, ele prevê a inclusão do protocolo *Openflow*, do projecto *openstack* entre outros, visto que os seus objectivos passam primeiro por estudar o mercado das *SDNs*, analisar o impacto da sua implementação na rede actual e permitir o desenvolvimento de uma solução normalizada que satisfaça as necessidades actuais do mundo das redes de computadores[20] [37].

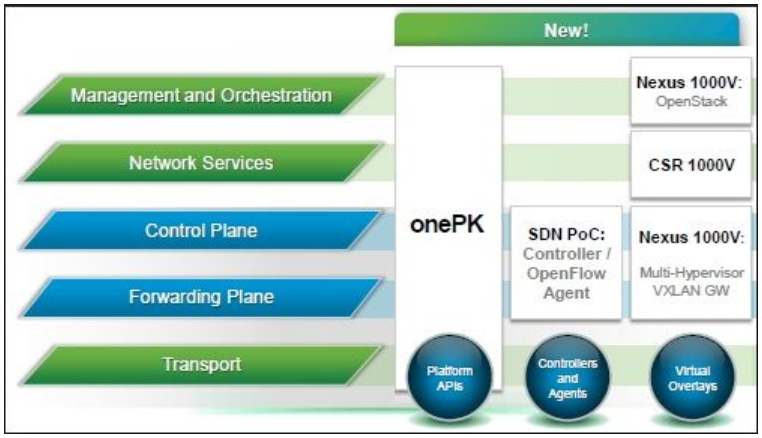

Figura 2.14: Solução *Openflow* da Cisco [43]

2.7.4 *OpenStack*

Finalmente temos o projecto *Openstack* [44], que corresponde a uma iniciativa de *software* livre para a criação de nuvens (*CC*). Entre os projectos mais importantes que apresentam temos o projecto *Quantum*,

cuja temática é a conectividade de rede, e que visa proporcionar uma ligação à rede como um serviço entre os dispositivos da interface física.

O Quantum corresponde a um nível de abstracção ao nível da aplicação de rede que se baseia na implementação de plug-ins para separar a conectividade física real da conectividade lógica. Devido às limitações da rede actual, propõe uma série de plug-ins que fornecem mais segurança, flexibilidade e qualidade de serviço mediante a criação de sub-redes privadas.

Um dos desafios lançados pela comunidade de *OpenStack* passa por oferecer de forma automatizada e rápida um sistema de servidores e de armazenamento de informação que não ofereça só um abstracção da rede, nem uma fraca flexibilidade. Desta forma tem-se intensificado a investigação sobre a utilização e adaptabilidade do *Openflow* para o projecto Quantum, existindo já os primeiros plug-ins referentes a estes temas. A prioridade passa por desenvolver um controlador de código aberto que permita a interligação entre o OpenStack e o Quantum [44].

2.7.5 Pertino

Pertino é uma pequena empresa de rede que ambiciona revolucionar o mundo do *CC* com a ajuda das *SDNs* para ser o catalizador das pequenas e médias empresas na aquisição de *hardware* e *software* a baixo custo, mas com maior flexibilidade e rentabilidade fornecidos pela arquitectura aberta e dinâmica das *SDNs*.

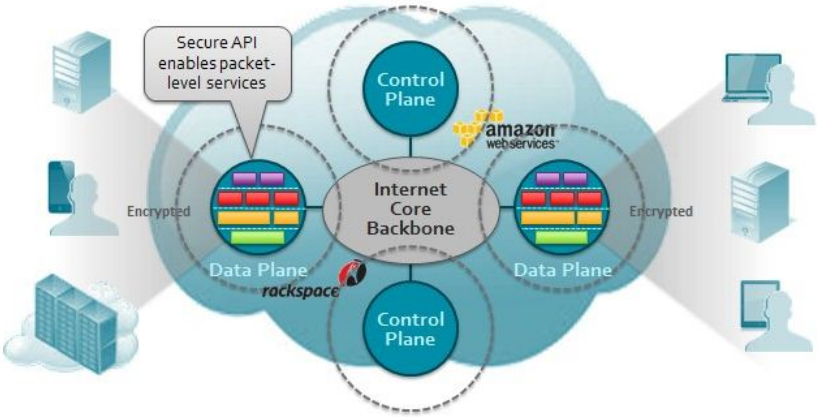

Figura 2.15: Solução *Openflow* da Pertino [27]

Pertino tem desenvolvido uma plataforma de *software* que combina o poder e omnipresença da *cloud* com a flexibilidade e universalidade das *SDNs* para a construção de uma rede completamente na *cloud* sem ter que recorrer ao *hardware*. Assim, Pertino permite a construção de uma rede, em poucos minutos, que liga a todos os utilizadores e os seus respectivos recursos em qualquer parte do mundo, deixando no passado a ideia das *Local Area Network (LAN)* e redes virtuais para gestão e manutenção da empresa. Em essência trata-se de uma rede *Network as a Service (NAAS)* como se pode observar na figura 2.15.

Entre os benefícios que fornece temos o acesso à informação em qualquer parte do mundo, acesso a aplicações de informática através de escritórios remotos, partilha de ficheiros, etc [27] [29].

2.8 Conclusão

A evolução tecnológica verificada nos últimos anos provocou um crescimento exponencial do número de dispositivos com acesso à Internet [23] [57], que deixaram em evidência as limitações e problemas da arquitectura actual da rede. A mudança dos padrões de tráfego, a pouca escalabilidade e falta de QoS são alguns dos problemas que constituem o fenómeno da ossificação da Internet. Este levou a comunidade científica a procurar uma solução a fim de satisfazer os requisitos dos utilizadores actuais, surgindo assim as *SDNs*.

As *SDNs* representam um novo paradigma no mundo das redes informáticas, apresentando-se como a alternativa de futuro, dadas as características da sua arquitectura, para uma rede mais dinâmica onde as aplicações usufruem de padrões de tráfego entre servidores, deixando no passado as aplicações cliente-servidor.

A abstração possibilitada pelo modelo da sua arquitectura, isto é, a separação integral do plano de dados e de controlo, centralizando o controlo num dispositivo numa camada superior da camada de infra-estrutuctura, oferece uma maior acessibilidade, flexibilidade e escalabilidade da rede. Com esta funcionalidade, passamos a ter uma visão geral da rede, isto é, a rede como um todo facilitando assim a gestão da rede em simultâneo com um aumento da qualidade de serviço. Por outro lado, a separação dos planos permite a reutilizaçao de equipamento existente a baixo custo.

O protocolo normalizado utilizado para criação, aplicação, manuntenção e gestão das *SDNs* é o *Openflow*, que pela sua potencialidade e o facto de ser uma fonte de código aberto, teve o apoio e atenção dos investigadores e empresas melhor cotadas a nível mundial. Contudo, o desfasamento entre a velocidade com que evolui a especificação e a que se desenvolve a tecnologia, quer a nível de controladores quer a nível de *hardware*, torna-se a sua principal dificuldade.

Contudo, as *SDNs* apresentam ainda umas quantas barreiras a vencer antes de se tornarem na nova arquitectura utilizada pela Internet, tais como o desenvolvimento de novas plataformas de *software* aberto de maior densidade que permitam aumentar o desempenho das implementações actuais e o desenvolvimento de políticas de segurança.

Por isso no próximo capítulo apresentar-se-á uma protótipo de solução que utilizará a arquitectura das *SDNs* devido ao dinamismo e abstração fornecida pelas mesmas. O protocolo de comunicação utilizado será o *Openflow* pois é o protocolo normalizado e mais utilizado actualmente nas *SDNs*.

Capítulo 3

Arquitectura do Demonstrador de rede *Openflow*

3.1 Introdução

Este capítulo apresenta uma proposta de solução para o problema da arquitectura actual de rede, que pretende mostrar as vantagens advindas das funcionalidades e características do novo paradigma de rede, isto é, das *SDNs* e o protocolo normalizado *Openflow*. Este capítulo tem como objectivo apresentar a arquitectura e os diversos módulos que a constituem, assim como a interacção entre os mesmos.

O capítulo inicia com o conjunto de requisitos e especificações que a solução deve apresentar, seguindo-se com a apresentação da arquitectura do protótipo, e a correspondente descrição comportamental e funcional de cada um dos módulos que integram a solução. Finalmente será apresentada a comunicação, interfaces e protocolos utilizados para a comunicação inter- e intra-módulos.

3.2 Proposta de solução

3.2.1 Descrição da solução

Como se apresentou na secção 1.1, os problemas de dinamismo, escalabilidade da rede actual e o crescimento contínuo do número de utilizadores, fazem com que seja importante encontrar uma solução que melhore o desempenho da rede. Uma possível solução para estes problemas poderá encontrar-se nas *SDNs*, que sendo uma tecnologia emergente, requer o teste das suas potencialidades e funcionalidades.

Por isso, uma possível resposta aos problemas da arquitectura actual poderá passar pelo desenvolvimento de uma rede física baseada nas *SDNs* em conjunto com a integração do protocolo *Openflow*, que garantam a comunicação entre dispositivos e permitam a activação e fornecimento de serviços de rede semelhantes aos que a arquitectura actual apresenta. Também deve ter capacidade de efectuar a monitorização, gestão e manuntenção da rede ao nível de serviços e estado da mesma.

A fim de demonstrar as vantagens e benefícios das *SDNs* e protocolo *Openflow* para o problema da ossificação de rede, propõe-se o desenvolvimento de um demonstrador *Openflow* que verifique as especificações acima explicadas, cuja arquitectura é apresentada de seguida.

3.2.2 Arquitectura da solução

O demonstrador desenvolvido apresenta uma arquitectura *SDNs*, como ilustra a figura 3.1, utilizando a camada de dados e camada de aplicação, tendo sido desenvolvido um gestor de rede na camada de controlo a fim de corresponder aos requisitos da rede actual.

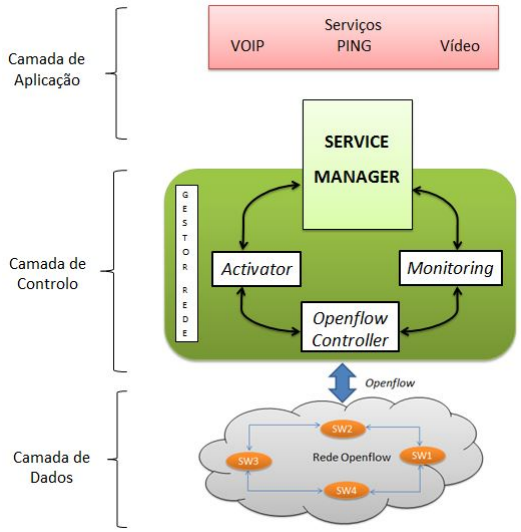

Figura 3.1: Arquitectura do Gestor de Redes

O módulo de serviço que se encontra inserido na camada de aplicação corresponde a um conjunto de módulos de alto nível responsáveis pela expedição de pedidos de activação de serviços na rede. Este módulo não configura nada na rede, só permite que os serviços estejam disponíveis para utilização dos clientes.

O gestor de rede localiza-se na camada de controlo da arquitectura *SDN*, e tem como funções a monitorização, manutenção e activação de serviços na rede, provenientes da camada imediatamente superior. Internamente este módulo contará com quatro sub-módulos responsáveis por garantir as especificações previstas na secção 3.2.1. O seu modo de funcionamento consiste em receber um pedido de activação de serviços de conectividade, proveniente do módulo de serviços, mediante a utilização da API. Depois é efectuada a análise e tratamento do mesmo para proceder à sua activação na rede, tendo em atenção o estado da rede em termos de serviços activos e topologia do substrato. Finalmente é efectuada a tradução dos pedidos de activação de serviços em *flows* para proceder à activação do serviço na rede, mediante a utilização do protocolo *Openflow*.

Por fim encontra-se o substrato da rede, que se insere na camada de dados, no qual se pretende por em prática a solução. O substrato é constituído principalmente por *switches*, *routers* e outros elementos de rede. Este módulo recebe em cada um dos seus elementos mensagens *Openflow* provenientes do gestor de rede, para a configuração dos dispositivos de acordo com os requisitos da rede.

Esta arquitectura fornece à solução independência entre módulos, o que aumenta a escalabilidade e flexibilidade da rede, permitindo integração, desenvolvimento e normalização de novas tecnologias e ferramentas para satisfazer as necessidades do cliente de rede, beneficiando a inovação de novos produtos e serviços. Outra das vantagens que advém desta arquitectura prende-se com o facto da centralização do plano de controlo permitir que a camada de controlo tenha uma visão completa da rede, possibilitando a sua alteração em tempo real de

acordo com os requisitos ou problemas da rede num dado instante.

A secção seguinte detalha a arquitectura do gestor de rede implementado, que representa a camada de controlo.

3.3 Gestor de Rede

3.3.1 Arquitectura e funcionalidades

O gestor de rede será constituído por 4 módulos, como ilustra a figura: 3.2, *monitoring*, *service manager*, *activator* e o *Openflow controller*.

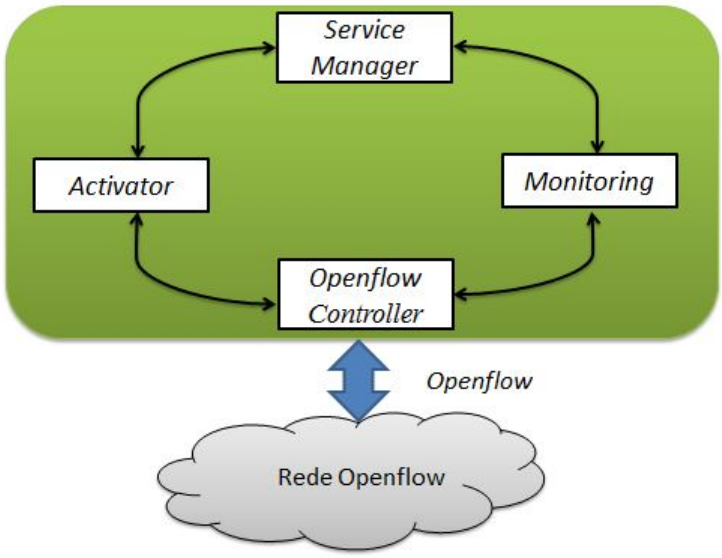

Figura 3.2: Arquitectura Interna do Gestor de Redes

3.3.1.1 *Service Manager*

O módulo *Service Manager* corresponde ao módulo mais inteligente da arquitectura apresentada na figura 3.2 e tem a responsabilidade de tomar decisões ao nível do mapeamento de fluxos. A sua arquitectura interna é apresentada na figura 3.3, onde se pode observar que é subdividido em dois módulos: o *Service Handler* e o *Flow Handler*. O primeiro é responsável por receber os serviços de conectividade que se pretende activar na rede, validar a informação do pedido e armazenar numa estrutua de dados interna. O *Flow Handler* é o responsável por decompor os pedidos em fluxos para os mesmos serem activados na rede mediante interação com o módulo *activator*. Assim sendo, podemos considerar que este módulo encontra-se na fronteira entre a

camada de aplicação e a camada de controlo, pois o *Service Handler* corresponde a um sub-módulo do mesmo que faz parte da camada de aplicação, já que o seu funcionamento prende-se com a disponibilização de serviços, enquanto que o *Flow Handler* se localiza na camada de controlo e é o responsável por realizar o pedido de activação do serviço a rede.

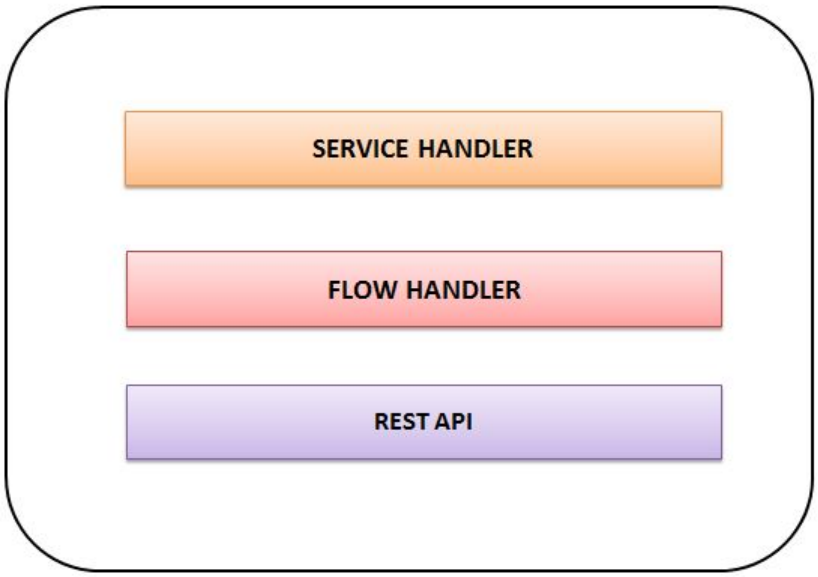

Figura 3.3: Arquitectura Interna do módulo *Service Manager*

De uma forma geral, as funcionalidades deste módulo são receber pedidos de activação de serviços, traduzir os pedidos de serviços em fluxos, efectuar os pedidos de activação dos fluxos na rede e reoptimizar a rede perante mudanças no substrato ou devido ao congestionamento da mesma, de acordo com uma determinada política interna que se adeqúe aos serviços em questão. Por fim toda a informação dos pedidos e dos fluxos a activar é armazenado numa estrutura de dados interna.

A fim de garantir o correcto funcionamento do módulo e a sua interação com os restantes módulos foram desenvolvidos os seguintes *URLs*, presentes na tabela 3.1 para aceder aos recursos que o módulo disponibiliza. Este módulo foi desenvolvido por uma Dissertação realizada em paralelo a esta Dissertação.

3.3.1.2 *Monitoring*

O módulo *Monitoring* é o responsável pela recolha e armazenamento da informação relativamente ao estado da rede e componentes do sistema. Tem como objectivos disponibilizar a informação a entidades imediatamente superiores e realizar o armazenamento da informação numa estrutura de dados interna. A sua a arquitectura interna é apresentada na secção 4.3.1, orientada ao serviço, oferecendo o serviço de topologia, alarme e informação estatística.

Entre as funcionalidades do módulo *monitoring* temos o levantamento e identificação de todos os elementos na rede (*links*, *switches*, e hosts), a construção da topologia física e o fornecimento de informação sobre a largura de banda dos *links*, bem como da informação estatística da rede. Outra das suas funções passa por realizar a análise dos parâmetros estatísticos do substrato, para obter conclusões sobre o estado da rede que permitam

URLs disponíveis pelo módulo Service Manager		
URL	**Verb HTTP**	**Descrição**
/mapping	POST	Recebe um pedido de activação de um serviço e devolve o *ID* correspondente.
/mapping/requestid	GET	Devolve a informação do pedido de activação do serviço identificado pelo request *ID*.
/mapping/requestid	PUT	Recebe uma actualização do pedido de activação do serviço identificado pelo *request ID*.
/mapping/requestid	DELETE	Elimina o pedido de activação do serviço identificado pelo *request ID*.
/topology	PUT	Recebe a alteração de topologia da rede.
/alarms	POST	Recebe a notificação sobre a existência de problemas na rede.

Tabela 3.1: Tabela dos *URLs* do *Service Manager*

detectar problemas e se traduzam na activação de alarmes para notificar o módulo imediatamente superior. Toda a informação é armazenada numa estrutura de dados.

3.3.1.3 *Activator*

O módulo *Activator* é responsável pela atribuição de fluxos na rede *Openflow*. Também fornece aos módulos superiores as informações das regras existentes no substrato e as suas topologias. Estes dados são guardados numa estrutura de dados interna de forma a facilitar o trabalho de identificação de fluxos. As suas decisões são directamente reflectidas no controlador. Tem como funções a atribuição e gestão de regras que controlam os fluxos no substrato, e o armazenamento das topologias relativas aos diferentes fluxos dos serviços a activar na rede. As suas decisões são reflectidas no *Openflow controller*.

3.3.1.4 *Openflow Controller*

O módulo *Openflow Controller* é o elo de ligação entre o plano de controlo e o plano de dados, encarregue de garantir a comunicação entre estas duas camadas mediante um canal seguro pelo qual existe troca de mensagens *Openflow*.

De acordo com as características referidas em 2.6.3.7 e 2.6.3.9, o *Openflow controller* escolhido foi o *Floodlight* que, como vimos nas ditas secções, encontra-se escrito na linguagem de programação *Java* e apresenta uma arquitectura orientada ao serviço, como a figura 3.4.

Observando a figura, pode-se dividir a arquitectura do *Floodlight* em três grandes camadas: a camada relativa à arquitectura interna do controlador, a qual é constituída por um total de 17 módulos; a camada de aplicações, constituída por sete módulos e finalmente a camada de comunicação constituída pela *REST API*.

Entre os módulos que fazem parte da arquitectura interna do controlador encontram-se: o *Module Manager* que é o responsável por fornecer e gerir a QoS na rede, mediante o desenvolvimento de *scripts* que implementam as políticas de policiamento na rede; o módulo *Thread Pool*, o qual oferece a capacidade ao *Floodlight* de correr vários *threads* em simultâneo; o *Jython Server*, que permite ao utilizador compilar módulos desenvolvidos pelos *developers* em *Jython*; a *Web Ui* corresponde a um portal de rede, onde o utilizador pode consultar toda a informação da mesma; O módulo *Unit Tests* é um módulo recente que disponibiliza um conjunto de testes para analisar o desempenho e funcionamento do módulo de *Firewall*, o qual será explicado mais adiante; o

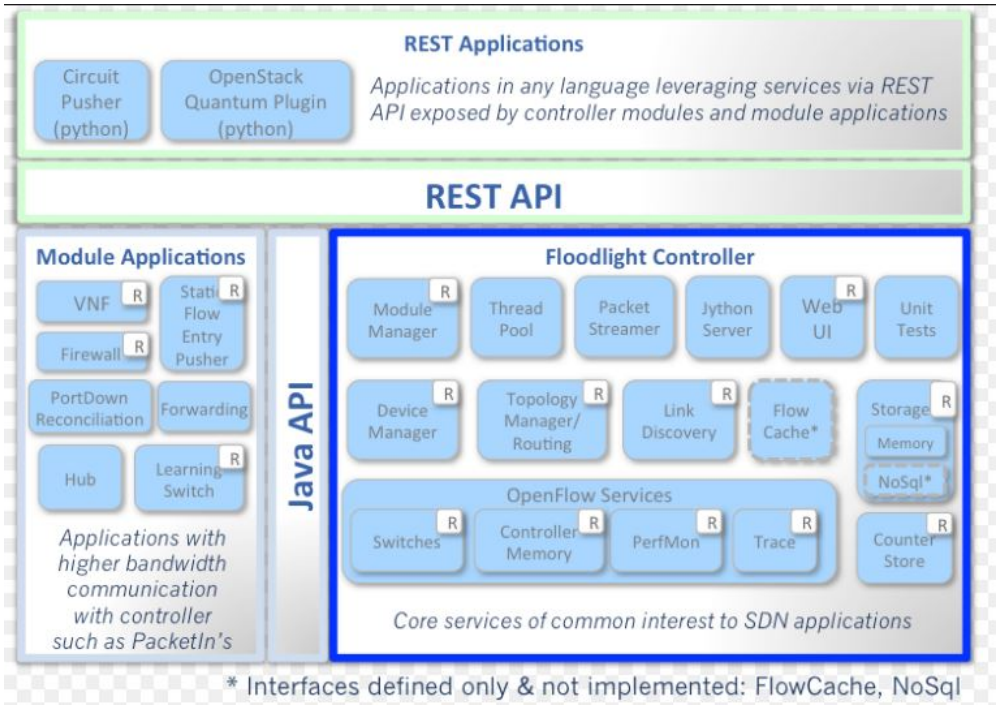

Figura 3.4: Floodlight1 [64]

módulo *Topology Manager* é o responsável pela topologia e mapeamento da rede, recorrendo ao algoritmo de Dijsktra; o *Link Discovery* está encarregue do estabelecimento de *links* na rede; o *Device Manager* tem como funcionalidade incluir novos nós na rede e fazer a respectiva actualização da topologia; o *Storage Source*: Módulo responsável por permitir o acesso à totalidade do plano de dados e reportar as suas alterações e actualizações; o módulo *switch* é o responsável pela adição, gestão e eliminação dos *switches* da rede *Openflow*; o módulo *Controller Memory* disponibiliza a informação sobre a memória dinâmica do controlador, de acordo com o número de *switches* e o número de fluxos presentes em cada um dos dispositivos. O *PerfMon* corresponde a uma bancada de testes para *scripts* e módulos desenvolvidos pelo utilizador de acordo com os requisitos das respectivas redes; o *PacketStreamer* é um serviço de transmissão de pacotes entre o *switch* e o controlador.

Relativamente à camada de aplicações tem-se o módulo *Forwarding*. Este realiza a inclusão de novos fluxos de *routing* extremo a extremo, para garantir a comunicação entre dispositivos, fornecendo a abstracção do controlo de *routing*. O *StaticFlowPusher* suporta a adição e remoção de fluxos estáticos. O *Virtual Network Filter* permite criar redes lógicas da camada *L2* numa única camada, e pode ser implementado como aplicação para *OpenStack* ou para utilização independente. O módulo *Firewall* define um conjunto de condições que permitem ou negam o tráfego de um determinado fluxo na rede. O *PortDown Reconciliation* foi desenvolvido para remover os fluxos que passem por uma interface ou porto que deixasse de funcionar.

Finalmente temos o *Rest Server*, servidor de reserva, que mantém as tabelas de encaminhamento e informações relevantes na gestão e manutenção de rede no caso de falha, e que permite a comunicação entre o controlador e gestor de rede.

O conjunto de *URLs* disponibilizados pelo *Rest Server* do *Floodlight* que vão ser utilizados se encontram presente na tabela 3.2.

A razão pela qual se escolheu o *Openflow controller Floodlight* prende-se com as suas funcionalidades,

URLs disponíveis pelo módulo *Floodlight*		
URL	**Verb HTTP**	**Descrição**
/wm/core/controller/switches /json	GET	Apresenta a lista de *MAC Address* dos *switches* presentes na rede.
/wm/core/switch/switchId /statType/json	GET	Fornece as estatísticas do *switch* identificado pelo campo *switchId*; o campo *statType* pode tomar os seguintes valores: *port, queue, flow, aggregate, desc, table, features, hosts.*
/wm/core/memory/json	GET	Apresenta a memória utilizada pelo controlador.
/wm/staticflowentrypusher/json	POST	Recebe um pedido de activação de um fluxo na rede *Openflow.*
/wm/staticflowentrypusher/list /switchId/json	GET	Apresenta a lista de fluxos activos no *switch* indicado pelo *switchId.*
/wm/staticflowentrypusher/list/all/json	GET	Apresenta a lista de todos os fluxos activos na rede.
/wm/staticflowentrypusher/json	DELETE	Elimina o fluxo indicado no interior da mensagem da rede

Tabela 3.2: Tabela dos URLs do *Floodlight*

anteriormente descritas, que fornecem ao gestor de rede uma abstracção maior dos módulos, o que se traduz numa melhor e mais eficaz manutenção da rede. O *Floodlight* tem uma *REST API* que aceita *scripts* em *Python*, com o objectivo de permitir a interacção entre o gestor de rede e a rede *Openflow*, facilitando assim a configuração, gestão e manutenção da rede. Outra das vantagens do *Floodlight* é o facto do *Floodlight* se encontrar no tipo de controladores que visa servir de transição entre os controladores centralizados, como por exemplo o *Nox*, e os controladores distribuídos, como por exemplo o*flowvisor*, pois permite as duas formas de utilização, traduzindo-se em maiores graus de liberdade nas aplicações e funcionalidades em que se pretenda testar e utilizar o *Floodlight.* O *Floodlight* apresenta uma equipa de trabalho muito coesa, versátil e diversa, que lhe permite oferecer uma *mailing list* sempre disponível para o esclarecimento de dúvidas sobre o funcionamento do controlador, assim como da especificação de *Openflow.* Esta última característica foi determinante no momento de escolha do controlador, pois são tecnologias emergentes que se encontram em fase de desenvolvimento e que permitem ter a oportunidade de interagir com outras pessoas que se encontram a trabalhar na mesma área da presente Dissertação. Foi utilizada a versão 0.9 de *Floodlight.*

3.4 Rede *Openflow*

Finalmente, para emular o substrato da rede, que se engloba na camada de dados, no qual pretende-se por em prática a nossa solução recorreu-se ao *OpenVswitch*, o qual será explicado nos seguintes parágrafos.

OpenVswitch é uma aplicação *open source* utilizada para realizar a comutação da pilha das camadas do modelo actual de rede. Foi concebido principalmente, para funcionar como *switch* virtual em ambientes virtualizados. Este *switch* fornece a visibilidade e controlo obtidos por um *switch* real, porém aproveita a flexibilidade prevista do *software* e da virtualização para realizar o controlo e gestão da rede [50] [51] [49] [10] [6].

O núcleo do *OpenVswitch* é constituído por duas entidades diferentes como mostra a figura 3.5: o *openvswitch server* e *openvswitch datapath.* O *openvswitch datapath* é responsável pela comutação dos pacotes que chegam ao *switch* e a sua gestão na tabela de encaminhamento. O *openvswitch server* é responsável

pela implementação da lógica dos pacotes, isto é, está encarregue pelo controlo da trajectória dos dados e as acções a associar a cada um dos pacotes que chegam ao *OpenVswitch* [51].

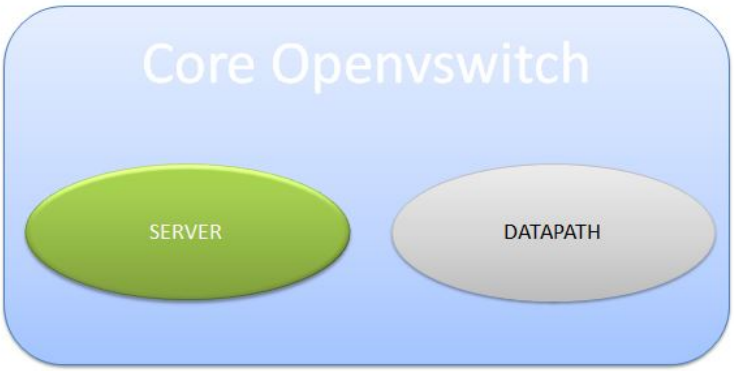

Figura 3.5: Arquitectura interna de um OpenVswitch

Openvswitch é um *switch* de *software* que reside no domínio de gestão dos ambientes virtualizados e proporciona conectividade entre as diversas máquinas virtuais e físicas presentes na rede. O seu funcionamento é semelhante ao apresentado por um *switch* básico real, da camada *L2*, contudo exporta um conjunto de interfaces e funcionalidades que permitem alterar o estado da rede em tempo real com o objectivo de aumentar o desempenho da mesma [50].

Interface de configuração: é a interface que permite ler e escrever a configuração e estado da rede, relativamente a factores tais como a topologia, portos entre outros. O gestor pode estabelecer a ocorrência de eventos para receber notificações da alteração do estado da rede em relação a um ou mais parâmetros. Também pode integrar aplicações externas, tais como o *NetFlow* e *sFlow*, para realizar a gestão da mesma, que lhe permitam melhorar o desempenho da rede. Outra das suas funcionalidades é efectuar a interligação entre os portos do *switch* e os do ambiente virtualizado em questão [49] [10].

Forwarding Path: é a interface que permite manipular remotamente a tabela de encaminhamento do *switch*. Desta forma o gestor de rede, utilizando um *script* ou processo externo, pode reescrever a tabela de encaminhamento do *OpenVswitch*, em tempo real, especificando como serão reencaminhados os pacotes na rede [51] [10].

Management Interface: esta interface permite gerir a configuração topológica da rede, isto é, permite a criação de múltiplos *switches*, gere a conectividade entre os mesmos e todavia permite o reenvio de tráfego baseado em regras, semelhante à tabela de fluxos utilizada nas redes *Openflow*. Desta maneira, as funções de estado e reenvio de configuração da rede podem estar associadas a um sub-conjunto de tráfego por um ou mais nós, considerando que a rede pode suportar ou não um determinado tráfego na rede.

Entre as funcionalidades que apresenta o *OpenVswitch* vale a pena destacar a possibilidade de integração de *software* de monitorização de rede, como por exemplo, o *Netflow* e *sFlow*, a partir dos quais podem-se detectar problemas na rede. Também permite a integração de analisadores de portos, tais como, *SPAN* e *RSPAN* que permitem enviar uma cópia dos pacotes que passam na rede para um porto de um outro *OpenVswitch* ou rede para realizar a gestão e manutenção da rede. Permite também a definição de VLANs, compatibilidade com o protocolo de rede *spanning tree*, permite o balanceamento do tráfego a passar no *switch*, para um *switch* real utiliza a ferramenta *Lateral Assessment Certification Program (LACP)* e para um *switch* virtual utiliza o *bonding*. Outra das suas funcionalidades é permitir a definição de níveis de QoS, ao nível da largura de banda,

latência entre outros parâmetros para melhorar o desempenho desta. Uma das principais funcionalidades para a solução proposta passa por permitir a integração de soluções de programação e funções de gestão da rede orientadas ao protocolo *Openflow* que permitem melhorar o seu desempenho [50] [49].

O *OpenVswitch* encontra-se disponível nos repositórios das actuais versões de *linux*, mas também pode ser utilizado em outros sistemas operativos, mediante uma correcta configuração das ferramentas necessárias. Para a solução proposta utilizar-se-á a versão 1.7.3 do *OpenVswitch* pois era a versão mais normalizada aquando do desenvolvimento da solução.

Em resumo, os módulos *activator* e *monitoring* estão em contacto contínuo com o *Openflow controller* com o objectivo de obter e proporcionar informação da rede *Openflow*. Por outro lado, o *service manager* utiliza a informação exposta pelo módulo de *monitoring* a fim de tornar a rede mais programável e independente da infra-estrutura.

A troca de informação entre módulos é realizada mediante *REST API* que será explicado na secção seguinte.

3.5 Comunicação entre módulos

A comunicação entre os módulos que constituem a arquitectura do gestor de rede será assegurada por quatro *RESTFul API*, desenvolvidas em linguagem *Python*, e pelas ferramentas associadas para este tipo de aplicações, que serão explicadas na secção seguinte. Antes de abordar as especificações dos módulos de comunicação e tipo de mensagens utilizadas, realizaremos um pequeno estudo dos conceitos adjacentes à arquitectura *REST*.

Ao longo das secções seguintes, e principalmente nos próximos capítulos, dar-se-á maior enfoque aos módulos *monitoring* e *activator*, pois foram os módulos desenvolvidos no âmbito desta Dissertação, tendo os restantes módulos externos contribuído para o desenvolvimento da arquitectura completa.

3.5.1 *REST*

O termo *REST* tem a sua origem aproximadamente no ano 2000, no trabalho doutural de Roy Thomas Fielding que define a *REST* como um estilo de abstração dos elementos arquitectónicos dentro dos sistemas distribuídos [54]. Desde esta perspectiva, pode-se definir *REST* como uma técnica de engenharia de *software* para a construção de aplicações distribuídas, inspirada nas características da web. Como esta permite aos clientes efectuar petições de serviços, é também considerada como uma arquitectura cliente-servidor que fornece serviços [53] [18].

A fim de compreender melhor o conceito de uma *REST* proceder-se-à a apresentação dos elementos que a constituem: os recursos correspondem às fontes de informação específica que pode ser acedida publicamente pelo seu identificador único e global (*URL*). O *URL* corresponde ao identificador global de recurso, pelo qual o cliente obtém uma representação do recurso. Estes devem manter uma hierarquia lógica.

Um dos princípios que define a *REST* é o funcionamento numa arquitectura cliente-servidor, onde os clientes solicitam ao servidor a activação ou informação de um dado serviço. A interação entre o cliente e o servidor é *stateless*, isto é, cada pedido do cliente ao servidor deve conter toda a informação necessária para compreender o pedido. Desta forma, qualquer servidor disponível pode dar resposta ao pedido feito pelo cliente. Outro princípio é a utilização de uma interface uniforme, em que as operações disponíveis sobre os recursos são sempre as mesmas, apresentando uma semântica normalizada e conhecida por todos os clientes e serviços. Finalmente, outro princípio *REST* importante é o facto de ser um sistema por camada, o qual implica que um elemento não pode ver mais além da camada imediatamente a seguir com a que interage, limitando desta forma a complexidade do sistema em geral e provomendo a independência entre camadas [54] [21].

A *REST* tem como objectivo a construção de sistemas distribuídos, constituídos por milhares de serviços desenvolvidos independentemente, escalando aos níveis comparáveis da web, ao se concentrar em minimizar o acoplamento entre serviços. A minimização da junção entre serviços consegue-se mediante a definição dos recursos e *URL* [17].

A *REST* apresenta uma elevada influência do protocolo *HTTP*, por tal motivo as acções sobre os recursos são baseadas no protocolo *HTTP* [18][17] [53]. As operações que permitem manipular o estado público de um recurso podem ser de 4 tipos:

- *GET*: com este comando o cliente obtém a representação do recurso identificado pela *URL*;
- *POST*: o cliente solicita ao servidor na petição a criação de um novo recurso. O servidor responde com o *ID* do novo recurso;
- *PUT*: como o servidor e o cliente têm uma cópia diferente do estado, o cliente pode, mediante esta operação, pedir uma cópia actualizada do estado actual do recurso;
- *DELETE*: o cliente solicita a eliminação de um recurso do servidor, sendo necessário que o cliente saiba o *ID* do elemento que pretende eliminar;

Os métodos *PUT e DELETE* funcionam sobre recursos *HTTP* pelo que é necessário ter muita precaução na utilização destes métodos para evitar a perda de informação crítica do sistema *RESTFul*.

Assim podemos definir que a *RESTFul* é o estilo de arquitectura orientada ao recurso, que apresenta uma API de acordo com os princípios *REST*. O seu método de funcionamento baseia-se sob recursos para permitir a interação entre o cliente e um servidor, que procuram aumentar a escalabilidade de interações entre componentes, fornecer segurança e encapsular os sistemas utilizados. A implementação mais comum é sobre *HTTP*, que como protocolo normalizado da Internet na comunicação permitiu a criação de outras tecnologias [53].

Depois de apresentar a arquitectura que permite a comunicação entre módulos, estudar-se-á a semântica e formato das mensagens que os módulos utilizam mediante a sua *REST API*, para proceder à troca de informação.

3.5.2 *Java Script Object Notation (JSON)*

A troca de informação entre os módulos efectuar-se-á por troca de mensagens entre as *REST API* desenvolvidas em cada um dos módulos. As mensagens serão formatadas em *JSON* [26], devido à sua facilidade no tratamento de informação e à sua elevada utilização no mundo das comunicações das redes informáticas.

JSON: é um formato leve de troca de informação entre dispositivos, baseada num sub-conjunto de especificações *JAVAScript*. Permite repesentar os dados em duas estruturas de dados: lista ordenada de valores, que na maioria das linguagens de programação corresponde a um *array*, vector ou lista, e colecções *key-value* que na maioria das linguagens de programação são caracterizadas por um objecto, dicionário ou *array*. A combinação de estes dois tipos básicos de estrutura representa qualquer informação independente da sua complexidade.

A fim de garantir fiabilidade e a consistência das mensagens escolheu-se a segunda forma de representar os dados, isto é, a forma de *key-value* que apresenta um comportamento semelhante aos dicionários em *Python*, nos quais a informação contida no seu interior é acedida por recurso à sua chave.

A semântica das mensagens JSON com esta estrutura consiste num conjunto de *keys-values*, que começa por uma chave de abertura e termina com uma chave de fecho. Cada *key* é seguido por ":" e os pares *keys-values* são separados por ",".

Depois de apresentar a arquitectura e a semântica por detrás da comunicação apresenter-se-á o tipo de mensagens utilizadas entre os módulos.

3.5.3 Comunicação entre o *Monitoring-Floodlight*

A comunicação entre o módulo *monitoring* e o *Floodlight* é baseada no conjunto de *URLs* presentes na tabela 3.2 que disponibilizam a informação para as diversas situações, as quais serão estudadas no capítulo seguinte.

3.5.4 Comunicação entre o *Service Manager-Monitoring*

O módulo *Service Manager* comunica com o módulo *Monitoring* mediante as *REST API* desenvolvidas e com base nas seguintes mensagens pré-definidas, que utilizam os *URLs* do módulo *Monitoring* apresentados na tabela 4.2:

- Mensagem de subscrição: corresponde à mensagem enviada em formato JSON, por parte do módulo externo, por exemplo o *Service Manager*, para o módulo de *Monitoring*, a realizar um pedido de activação de subscrição por parte do(s) cliente(s), referente a informação sobre a disposição do substrato da rede e possíveis alterações e problemas (alarmes) que se verifiquem na rede.
- Mensagem de alarme: corresponde à mensagem enviada pelo módulo *monitoring* para um módulo externo, que tenha efectuado correctamente a sua subscrição de alarme, no caso de detectar alguma anomalia na rede, por exemplo, a perda de um *link*, ou se a percentagem de perdas numa interface ultrapassar um determinado limiar. Esta mensagem resulta da contínua monitorização de todas as interfaces do substrato da rede.
- Mensagem de eliminação de subscrição: corresponde à mensagem enviada pelo módulo externo para o módulo *monitoring* a pedir a eliminação de subscrição de um dado cliente, identificado pelo seu *ID*.

3.5.5 Comunicação entre o *Service Manager-Activator*

O módulo *Service Manager* comunica com o módulo *Activator* mediante as *REST API* desenvolvidas e com base nas seguintes mensagens pré-definidas, que utilizam os *URLs* do módulo *Activator* apresentados na tabela 4.3

- Mensagem de activação de fluxos: corresponde à mensagem enviada pelo módulo externo, o *Service Manager* para o módulo *Activator*, a fim de permitir o fluxo de um determinado tipo de tráfego no substrato da rede com características especificadas no interior da mensagem JSON e de acordo com a política interna do módulo externo.
- Mensagem de actualização de fluxos: corresponde à mensagem enviada pelo módulo externo para o módulo *activator* a notificar a alteração da informação, ou parte dela, de um fluxo previamente activado que é identificado pelo seu *flow ID*.
- Mensagem de eliminação de fluxos: corresponde à mensagem enviada pelo módulo externo para o módulo *activator* a pedir a eliminação da rede de um determinado *flow*, que é identificado pelo seu *flow ID*, que activa um determinado tipo de tráfego no substrato.

3.5.6 Comunicação entre o *Activator-Floodlight*

A comunicação entre o módulo *Activator* e o *Floodlight* realizar-se-á pelos *URLs* presentes na tabela 3.2, para activação, actualização e eliminação dos fluxos, como será estudado no capítulo seguinte.

3.5.7 Comunicação entre o *Floodlight* e a rede *Openflow*

A comunicação entre o *Openflow controller*, no nosso caso o *Floodlight* e a rede *Openflow* é garantida pelo protocolo *Openflow* que foi estudado na secção 2.6.1. O protocolo *Openflow* possui três tipos de mensagens: *controller-switch*, assíncronas e simétricas.

A mensagem *controller-switch* inicia-se no controlador e tem como objectivo analisar o estado do *switch*. As mensagens assíncronas podem ser geradas pelo *switch* para comunicar ao *Openflow controller* de eventuais alterações na rede. As mensagens simétricas são mensagens trocadas entre o *Openflow controller* e o *switch* sem existir um pedido específico e servem para sincronização entre o *Openflow controller* e o *switch*. A mensagem assíncrona é enviada pelo *switch* sempre que se verifica o início de uma sessão de tráfego, isto é, sempre que começam a chegar novos pacotes correspondentes a um fluxo. Também são enviadas sempre que se verificar um erro na comunicação anterior, ou se se verificar a falha do *switch*.

A mensagem assíncrona pode ter origem quer no *Openflow controller* quer no *switch* e serve para sincronização da comunicação entre os mesmos.

A seguinte tabela visa resumir as principais mensagens trocadas entre o *Openflow controller* e o *switch*:

Mensagens entre o Controlador e rede *Openflow*		
Mensagem	**Tipo**	**Descrição**
Hello	Controlador->*Switch*	À semelhança do protocolo *TCP*, há uma troca de mensagem entre o controlador e *switch* para sincronizar as comunicaçoes entre os mesmos.
Features Request	Controlador->*Switch*	O controlador solicita ao *switch* os portos disponíveis para a comunicação.
Set Config	Controlador->*Switch*	O controlador solicita ao *switch* o envio dos tempos de cada fluxo.
Features Reply	*Switch*-> Controlador	O *switch* envia ao controlador a lista de portos disponíveis, com as suas velocidades.
Port Status	*Switch*->Controlador	O *switch* notifica ao controlador sobre alterações nas características dos portos.

Tabela 3.3: Mensagens entre o *Floodlight* e a rede *Openflow*

Mensagens entre o Controlador e rede *Openflow*		
Mensagem	**Tipo**	**Descrição**
Packet-In	*Switch*->Controlador	Mensagem utilizada pelo *switch* para notificar ao controlador da chegada de um pacote que não possui fluxo atribuído na tabela de fluxo do *switch*.
Packet-Out	Controlador->*Switch*	O controlador envia o pacote recebido no *packet-in* para um ou mais portos do *switch*.
Flow-Mod	Controlador->*Switch*	O controlador notifica o *switch* para adicionar um fluxo particular nas suas tabelas de fluxo.
Flow-Expired	*Switch*->Controlador	O *switch* pede ao controlador para remover um determinado fluxo, após verificado um período de inatividade.

Tabela 3.4: Mensagens entre o *Floodlight* e a rede *Openflow* (continuação)

3.6 Requisitos e Dependências de *Software*

Nesta secção abordam-se os requisitos necessários para o desenvolvimento e utilização de cada um dos módulos que constituem a solução.

Os módulos *Monitoring*, *Service Manager* e *Activator* foram módulos desenvolvidos na linguagem de programção *Python* por ser uma linguagem de alto nível, próxima da linguagem falada, de fácil acesso e *open source*. É uma linguagem muito utilizada no mundo da investigação. As dependências e requisitos para a utilização deste módulos são a versão 2.7 ou superior da linguagem, sendo necessário proceder à instalação do módulo *Bottle Web Server*, utilizado para o desenvolvimento das *REST API* e a biblioteca *httplib* para o estabelecimento da comunicação entre módulos.

Para o desenvolvimento do módulo da rede *Openflow* recorreu-se à ferramenta *OpenVswitch* na versão 1.7.3, a qual apresenta vários requisitos e dependências para o correcto funcionamento, entre os quais, a utilização do sistema operativo *linux* cuja versão de *kernel* não deve ultrapassar a 3.3. Nos repositórios deste sistema operativo é necessário incluir os seguintes pacotes, nas suas versões mais actualizadas, *GNU, automake, autoconf, perl*. Também é necessária a instalação de *Python* na sua versão 2.7 ou superior e das seguintes bibliotecas e ferramentas da linguagem PyQt4, Graphyz, *Python Twisted Concha*, *Python* JSON, *PySide* e *Python Zophe*.

3.7 Conclusão

Neste capítulo apresentou-se a arquitectura e os módulos adjacentes do protótipo de solução proposto neste projecto de Dissertação. Foram também descritos os módulos de forma comportamental e funcional e finalmente indicou-se o mecanismo de comunicação a utilizar pelos módulos.

Capítulo 4

Implementação do Demonstrador *Openflow*

4.1 Introdução

Este capítulo tem como objectivo descrever a implementação do protótipo da solução apresentada no capítulo anterior para a arquitectura do demonstrador *Openflow*. Começar-se-á por apresentar a *testbed* proposta, referindo as características e especificações para sua construção. Seguidamente serão apresentadas as arquitecturas e funcionalidades dos módulos desenvolvidos no presente trabalho, mais concretamente o módulo de *monitoring* e o módulo *Activator*.

4.2 *Testbed*

Devido às limitações a nível da tecnologia desenvolvida das *SDNs*, onde a especificação dos protocolos caminha a um ritmo superior ao qual a tecnologia e soluções de *software* conseguem dar resposta, propõe-se uma topologia de rede simples, mas adequada para as ferramentas e aplicações sólidas e disponíveis nos dias de hoje, que permitirão demonstrar as vantagens e o valor acrescentado da arquitectura das *SDNs* e em específico do protocolo *Openflow*.

A topologia física proposta é constituída por 4 *switches Openflow* emulados por *OpenVswitch* com recurso a 4 *servers* e 4 clientes, ver figura 4.1. Através desta topologia, que simulará uma rede real, pretende-se avaliar as vantagens das especificações das *SDNs* e do *Openflow*, tais como a separação de tráfego e reacção a saturação de *links* sem ser necessária a reconfiguração dos equipamentos, separando assim o plano de controlo do plano de dados. A interligação entre os nós da *testbed* foi realizada com a ajuda da *tag vlan* de acordo as especicações do protocolo 802.1Q

As características do *hardware* e *software* da *testbed* encontram-se resumidas na tabela 4.1. As especificações e versões utilizadas ao *software* adjacente a solução, tais como o *Floodlight*, *Openflow*, *OpenVswitch* foram estudados no capítulo 2 e 3 nas secções 2.6 e 3.3.1.

Os módulos desenvolvidos no âmbito deste trabalho, assim como os módulos externos que complementam e permitirão testar a solução, proposta no capítulo 3, encontram-se disponíveis no *switch* 1 como ilustra a figura 4.2

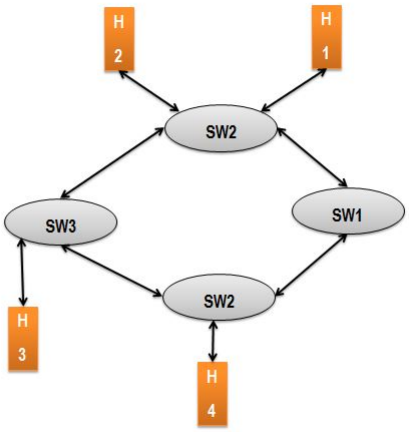

Figura 4.1: *Testbed* criada

Especificação *Testbed*				
Nó	**SW1**	**SW2**	**SW3**	**SW4**
CPU MODEL	Intel PentiumD 950	Intel PentiumD 950	Intel Core2 Duo 6400	Intel Core2 Duo 6400
CPU Freq	3.40GHz	3.40GHz	2.13GHz	2.13GHz
CPU Core	2	2	2	2
CPU Threads	4	4	2	2
HDD Memory	40GHz	40GHz	145GHz	145GHz
Random Access Memory (RAM) Amount	6	6	4	4
RAM Freq	667 MHz DDR2	667 MHz DDR2	533 MHz DDR2	533 MHz DDR2

Tabela 4.1: Especificação das características do *hardware* e *software* da *testbed*

Nas secções seguintes abordar-se-á em detalhe a criação e desenvolvimento do gestor de rede, apresentado na secção 3.3, dando maior ênfase aos módulos *monitoring* e *activator* da arquitectura apresentada na secção 3.3.1 pois foram os desenvolvidos no âmbito deste trabalho.

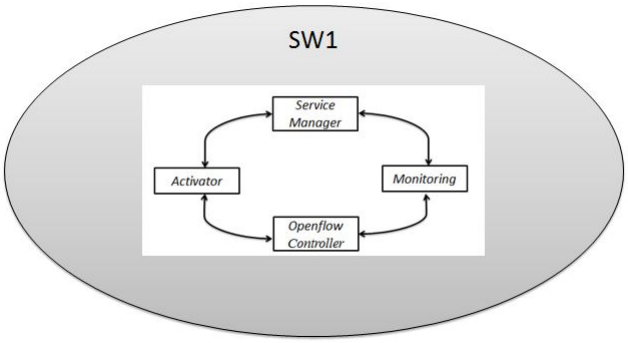

Figura 4.2: Localização dos módulos presentes na solução

4.3 *Monitoring*

4.3.1 Arquitectura

O módulo *monitoring* apresenta uma arquitectura interna orientada ao serviço e constituída por quatro sub-módulos: o de topologia, alarme, estatística e a *REST API*, como se pode observar na figura 4.3. Os serviços que o módulo oferece são o serviço de topologia, alarme e informação estatística da rede.

O método de funcionamento do módulo *monitoring* segue o princípio de funcionamento do modelo cliente-servidor: o cliente para receber a informação deve efectuar previamente a subscrição ao serviço.

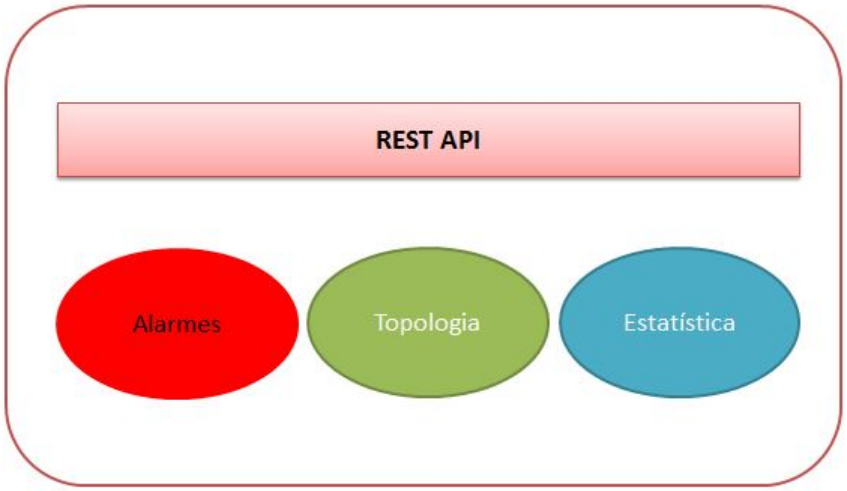

Figura 4.3: Arquitectura Interna do módulo *Monitoring*

O sub-módulo topologia é responsável por indicar a disposição dos nós da rede, a forma como se interligam, e apresentar a lista de portos por *switch* que estão activos. O sub-módulo alarme é o responsável por efectuar uma monitorização contínua da rede, e de acordo com a análise da informação recolhida, notificar possíveis

problemas, como por exemplo, a perda de um *link* ou aumento da percentagem da perdas de pacotes num porto. Finalmente, o sub-módulo estatística disponibiliza a informação estatística da rede, podendo esta ser apresentada por elemento de rede ou por interface.

4.3.2 Funcionalidades e Algoritmos

O módulo de *monitoring*, como foi referido na secção 3.3, é responsável pela monitorização, gestão e manutenção da rede.
Para a criação deste módulo optou-se pelo desenvolvimento de um conjunto de funções e *scripts*, escritos na linguagem de programação *Python*, que irão interagir com alguns dos recursos disponibilizados pelos *URLs* presentes na tabela 4.2 e na tabela 3.2, com o objectivo de verificar as especificações no âmbito do projecto.

4.3.2.1 Serviço de Topologia

A primeira das funcionalidades fornecidas por este módulo é o serviço de topologia. O serviço de topologia é garantido para todos os clientes que efectuarem a sua correcta subscrição, beneficiando assim da recepção da informação relativamente à disposição actual dos elementos de redes, a sua capacidade de largura de banda e a lista dos IDs dos elementos de rede. A informação será disponibilizada na forma de matriz adjacência.

O processo de subscrição da topologia é apresentado no fluxograma da figura 4.4.

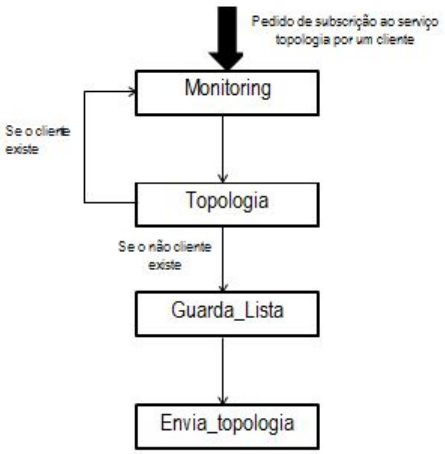

Figura 4.4: Fluxograma do serviço topologia

Os clientes que pretendam subscrever o serviço deverão efectuar o pedido *HTTP*, utilizando o *URL* correspondente (/subscription/topology), que se encontra definido na tabela 4.2.

A função *topologyform* é a função encarregue de receber o pedido de subscrição e verificar se o mesmo é correcto, isto é, se o pedido é realizado de acordo com as especificações de comunicação descritas no capítulo 3. Também tem como função verificar se o cliente que realiza a subscrição já não está subscrito ao serviço. A função *GuardaLista* é a função responsável pelo armazenamento da informação dos clientes que pretendem subscrever o serviço de topologia. Finalmente, a função *sendtopology* é a função responsável pelo envio da topologia física, a capacidade de largura de banda dos *links* e a disposição dos *switch*.

A obtenção da topologia física, a capacidade de largura de banda e a disposição dos *openvswitch* é garantida pelo *script topology*, que apresenta o fluxograma da figura 4.5.

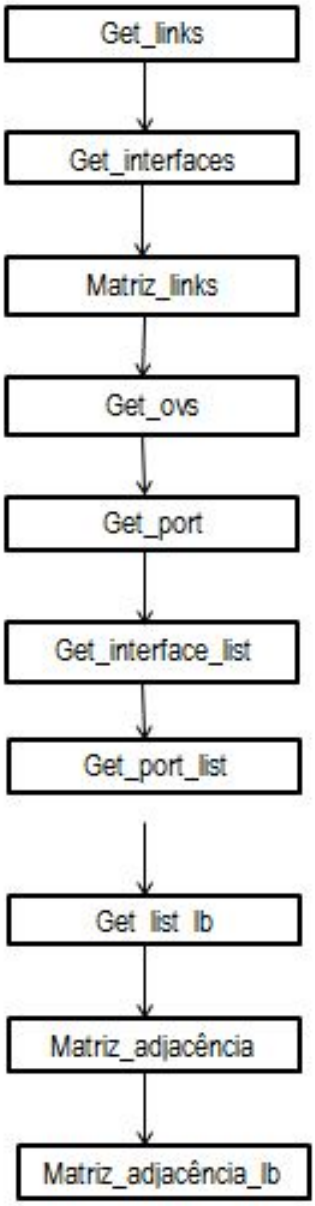

Figura 4.5: Fluxograma da função topologia

Por detrás destes dois grandes módulos que garantem o serviço da topologia, encontra-se um conjunto de funções de menor complexidade que ajudam no correcto funcionamento do serviço. De seguida apresentam-se as as mesmas.

- *Getlinks* é a função responsável por estabelecer a comunicação com o *Floodlight*, de acordo com os *URLs* disponibilizados pela sua *REST API*, que podemos observar na tabela 4.2, com o objectivo de obter a

informação dos *links* da rede, isto é, a forma como os *OpenVswitch* se interligam.

- *Getinterfaces* é função responsável por estabelecer a comunicação com o *Floodlight*, de acordo aos *URLs* disponibilizados pela sua RESTAPI, que podemos observar na tabela 4.2, e recolher a informação relativa às interfaces de cada um dos *OpenVswitch*.
- *Matrizlinks* é a função encarregue por fazer o tratamento da informação recolhida pela função Getlinks, disponibilizando-a na forma de lista e indicando que *OpenVswitch* estão directamente ligados.
- *Getovs* é a função que permite obter os endereços MAC address dos *OpenVswitch* presentes na rede *Openflow*.
- *Getport* é a função que tem como objectivo obter a lista das interfaces de cada um dos *OpenVswitch*, a partir do tratamento da informação da função *Getinterfaces*.
- *Getinterfacelist* é a função responsável por determinar e devolver a lista das ligações realizadas pelos *switches*, isto é, identifica a que tipo de elemento de rede o *switch* liga, ou seja, se é um *hosts* ou um *switch*.
- *Getlistlb* é a função que devolve a informação sobre as capacidades de largura de banda de cada uma das interfaces dos nós da rede.
- *MatrizAdjacencia* é a função responsável pela construção da matriz adjacente da topologia física da rede.
- *Matrizadjacencialb* é a função responsável pela construção da matriz das capacidades dos *links* da rede.

4.3.2.2 Subscrição ao serviço de alarme

A segunda funcionalidade fornecida pelo módulo *monitoring* é permitir a subscrição dos clientes com o objectivo de serem notificados de possíveis anomalias na rede *Openflow*.

A subscrição de alarmes é descrita por um fluxograma semelhante ao fluxograma da subscrição da topologia, como ilustra a figura 4.6

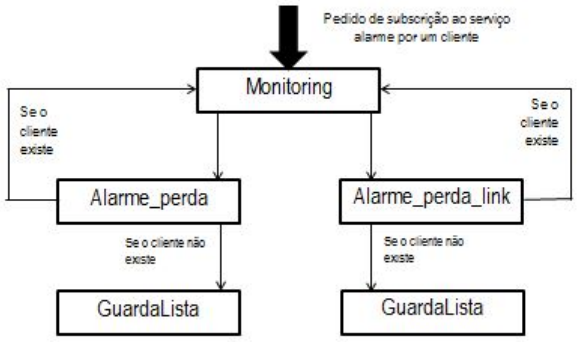

Figura 4.6: Fluxograma da subscrição do serviço Alarmes

Os clientes que pretendam subscrever o serviço deverão efectuar o pedido *HTTP*, utilizando o *URL* correspondente, que se encontra definido na tabela 4.2.

O funcionamento do módulo *alarmform* é o seguinte, como pode ser observado na figura 4.6. É realizado um pedido de subscrição por parte do cliente de acordo com as especificações e formato estudado no capítulo 3. O módulo GuardaLista verifica se o cliente se encontra já subscrito ao serviço de alarme: em caso positivo devolve-lhe o *ID* correspondente quando efectua a subscrição; em caso negativo armazena o novo cliente na base de dados desta lista de subscrição e fornece-lhe o seu *ID*.

4.3.2.3 Alarme Perda de *Link*

Como referimos anteriormente, uma das funcionalidades do módulo *monitoring* é enviar alarmes no caso de surgirem problemas na rede. Na versão actual o módulo é capaz de notificar a perda de *link* de alguns dos nós, isto é , *switch*, recorrendo a função *vetopology*. O fluxograma da função *vetopology* é apresentado na figura 4.7. O seu método de funcionamento consiste em verificar a topologia física da rede em dois instantes de tempos diferentes, comparando-as e no caso de serem diferentes, notifica a lista de clientes que efectuaram a subscrição ao serviço de topologia, como explicado na secção anterior. O *script* é executado em ciclo infinito com uma periodicidade de 45 ms.

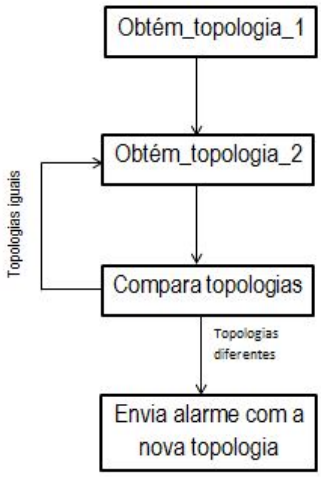

Figura 4.7: Fluxograma da alarme perda de *link*

4.3.2.4 Alarme percentagem de perdas

Outro dos problemas que a actual versão do módulo *monitoring* é capaz de notificar é se a percentagem da taxa de perda de algum dos *links* ultrapassa o limiar previsto através do módulo *alarmeperdas*.

O módulo *alarmeperdas* é apresentado no fluxograma da figura 4.8, e é responsável pela recolha periódica da informação estatística dos elementos de redes, fazendo comparação da mesma em dois instantes de tempo diferentes, calculando a percentagem de perdas. No caso em que a percentagem de pacotes ultrapasse um determinado limiar notificará os clientes que efectuaram a subscrição do serviço.

Neste *script* encontra-se um conjunto de funções de menor complexidade que ajudam ao seu desempenho e funcionamento, entre as quais se destacam:

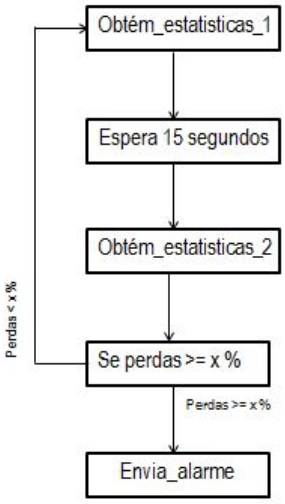

Figura 4.8: Fluxograma de alarme percentagem de perdas

- *Getstatistics* é a função responsável pela recolha da informação estatística proveniente do *Openflow controller*.
- *Calcula* é a função responsável por estimar a percentagem da taxa de perdas na comunicação entre switchs.

- *Sendalarm* é a função responsável pelo envio da informação dos alarmes gerada para os dispositivos de subscritores.

4.3.2.5 Fornecimento de informação estatística

Finalmente a última das funcionalidades do módulo *monitoring* passa pelo fornecimento da informação estatística da rede, tendo-se desenvolvido as seguintes funções para o seu funcionamento:

- *Matrizstatics* é a função responsável pelo tratamento da informação recolhida pela *Getstatistics* a qual é disponibilizada na forma de matriz.
- *FilterStatistics* é a função encarregue de efectuar a filtragem da informação recolhida e disponibilizada pelas funções *Getinterfaces* e a *Matrizstatistics*. Esta filtragem pode ser efectuada por *links* que ligam a cada *switch*, *hosts* ou à totalidade das interfaces.
- *ListswitchStatistics* função que devolve a informação da lista de *switches* que apresentam informação sobre *hosts*, switchs ou ambos.

4.3.3 *URLs*

A série de *URLs* definidos pelo módulo *monitoring* para os módulos superiores, a fim de fornecer um conjunto de serviços verificados na sua arquitectura, encontram-se defenidos na tabela 4.2. Para a representação da informação de cada um destes *URLs* são utilizadas as funções explicadas na secção 4.3.2. Assim sendo

os *URLs* que se encontram relacionados com a topologia, por exemplo, /subscription/topology utilizam as funções descritas na secção 4.3.2.1. Por outro lado os *URLs* que apresentam informação sobre as alarmes usam as funções 4.3.2.2. Finalmente os *URLs* que fornecem informação sobre as estatísticas utilizam as funções apresentadas na secção 4.3.2.5.

URLs do módulo *Monitoring*		
URL	**VERB HTTP**	**Descrição**
/stat	GET	Devolve os tipos de estatísticas para consulta do *switch*
/stat/stattag	GET	Devolve os *ID* dos *switches* associados à *stat tag*
/stat/stattag/switchid	GET	Devolve as estatisticas das interfaces associadas *stat tag* do *switch* com a identificação *switchID*
/subscription	GET	Devolve os tipos de subscrição possíveis
/subscription/alarms	GET	Devolve os tipos de alarmes possíveis
/flowsactive/switchid	GET	Devolve a lista de *flows* activos no *switchID*
/subscription/topology	POST	Subscripção do endereço para a recepção da topologia física e informação de largura de banda dos *links* (ambas na forma de matriz adjacente)
/subscription/alarms/ /alarmstag	POST	Subscripção do endereço para o alarme identificado pelo *alarms tag*
/subscription/topology/ /topologyid	DELETE	Elimina da lista de subscripções do endereço associado ao *ID* identificado pelo *topologyID*.
/subscription/alarms/ /alarmstag/alarmsid	DELETE	Elimina da lista de subscripções do endereço associado ao *ID* identificado pelo *ID* para o alarme identificado por *alarmsID*

Tabela 4.2: URLs do módulo *Monitoring*

4.4 *Activator*

O módulo *activator*, como foi referido no capítulo 3, é o encarregue pela activação de serviços provenientes da camada de aplicação que são tratados e reencaminhados pelo módulo *Service Manager* na rede.

A criação deste módulo implicou o desenvolvimento de um conjunto de funções e *scripts*, escritos na linguagem de programação *Python*, alguns deles incluídos na *RESTFulAPI*, a fim de garantir o correcto funcionamento da rede e verificar as especificações e objectivos previstos no âmbito deste trabalho.

Entre as funcionalidades que apresenta o módulo de activação encontram-se: a atribuição e gestão de regras que controlam o tráfego da rede e o armazenamento as diferentes topologias dos fluxos.

4.4.1 Funcionalidades e Algoritmos

4.4.1.1 Activação de um fluxo

A activação de um serviço na rede desencadeia o seguinte conjunto de acções que se pode observar no fluxograma da figura 4.9.

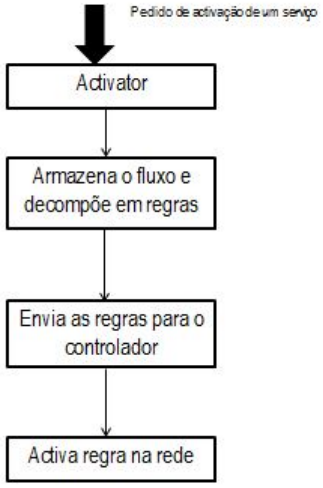

Figura 4.9: Fluxograma da activação de um serviço

Quando o pedido de activação de serviço chega ao módulo de activação, por meio dos *URLs* presentes na tabela 4.3, a primeira acção a ser realizada é a de verificar se o serviço em questão já não se encontra activo. No caso de já existir, devolver o *ID* atribuído quando foi activado o serviço. No caso em que o serviço não se encontra activo, procede-se à sua decomposição nos respectivos fluxos que serão armazenados na base de dados do módulo. Seguidamente é efectuada a tradução dos pedidos em regras a activar na rede de acordo com a topologia indicada. Finalmente, e consistente com a tabela dos *URLs* da *REST API* do *Floodlight* presente na tabela 3.2, procede-se ao envio das regras, correspondentes ao serviço a activar para o controlador, o qual se encarregará de efectuar a activação na rede.

Por detrás deste número elevado de módulos encontram-se as seguintes funções responsáveis por garantir o correcto funcionamento do módulo:

- *Getdata* é a função responsável por garantir que não são activados dois serviços iguais na rede. Devolve, no caso de um serviço repetido o *ID* aquando do momento de activação. Caso contrário atribui o novo *ID* ao serviço e devolve-ó.
- *Getname* é a função responsável por dar o nome da regra com o qual esta ficará activa na rede e na base de dados.
- *Gettopology* é a função responsável por fazer o armazenamento da topologia do serviço que se pretende activar.
- *Getswitchsporttopology* é a função responsável por fazer a tradução da topologia de *flows*, recebida no formato de matriz adjacente, nas ligações entre *switch*, indicando os seus portos, assim como nos *switches* que devem ser activadas as regras.
- *GetData2* é a função encarregue da decomposição e armazenamento dos serviços em fluxos, que se pretendem activar na rede. Por tal motivo pode ser considerada como o núcleo do *activator*.
- *SendtoController* é a função responsável pelo envio das regras em que foi decomposto o serviço para a

sua activação na rede, usufruindo dos *URLs* disponibilizados pela *REST API* do *Floodlight* de acordo com a tabela 3.2

4.4.1.2 Actualização de um fluxo

Outra das funcionalidades do módulo *Activator* é a de permitir a actualização de um fluxo. Tal pedido desencadeia as seguintes acções representadas no fluxograma da figura 4.10.

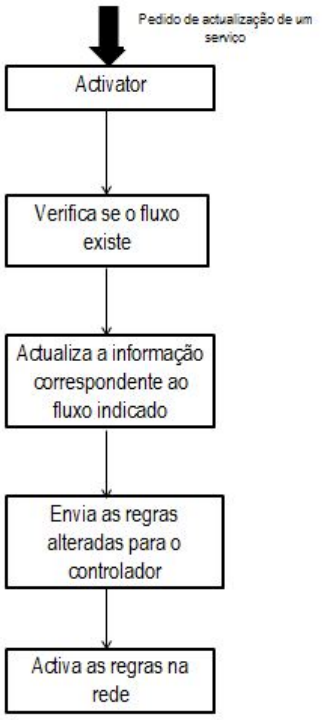

Figura 4.10: Fluxograma da actualização do serviço

Quando o pedido de actualização de um serviço chega ao módulo de activação, por meio dos *URLs* presentes na tabela 4.3, a primeira acção a ser realizada é a de verificar se o serviço indicado para actualizar se encontra activo; no caso de não estar activo, o módulo notifica esta informação. No caso em que o serviço exista efectua-se a sua actualização o que implica analisar o conjunto de regras activas na rede e correspondentes ao serviço que são afectadas pela actualização para proceder à sua actualização. Depois desta análise, o processo de actualização segue um comportamento semelhante à activação de um fluxo, isto é, procede-se à decomposição e armazenamento das novas características do serviço, efectua-se a sua tradução nas novas regras a activar, e finalmente, e de acordo com a tabela dos *URLs* da *REST API* do *Floodlight* presente na tabela 3.2, procede-se ao envio das regras correspondentes ao serviço a activar, para o controlador, o qual se encarregará de efectuar a activação na rede.

Mais uma vez, para o correcto funcionamento do fluxograma anterior foi implementado um conjunto de funções:

- *Putflow*: é a função encarregue de verificar que o *flow ID* do fluxo que se pretende activar existe e que

no caso de não existir devolve uma mensagem de erro.

No módulo actualizar desenvolveram-se as seguintes funções:

- *Getregrasafectadas* é a função responsável por conferir na base de dados todas as regras associadas ao serviço que se pretende actualizar, devolvendo esta informação na forma de lista.
- *Getinforegrasafectadas* é a função encarregue de comparar a informação dos fluxos pretendidos pela actualização com a informação dos fluxos afectados. No caso de serem diferentes assinala-os como fluxos a eliminar.
- *Deleteregras* é a função encarregue de eliminar da base de dados e da rede o conjunto de fluxos que já não são necessários após a actualização do pedido.
- *Sendputdata* é a função responsável pelo envio dos novos fluxos a activar na rede provocando uma actualização no serviço. Utiliza para tal os *URLs* disponibilizados pela *REST API* do *Floodlight* de acordo com a tabela 3.2.

4.4.1.3 Eliminação de um Serviço

Finalmente, a última das funcionalidades do módulo *activator* corresponde à capacidade de eliminar um serviço da rede. Para tal, utiliza a função *Deleteflow*, encarregue de verificar a existência do serviço na rede, e de seguida usufrui da função *Deleteregras* para eliminar as regras associadas ao serviço e notificar o controlador para depurar o serviço da rede.

4.4.2 *URLs* disponibilizados pelo módulo Activator

A série de *URLs* definidos pelo módulo *activator* para os módulos superiores, a fim de fornecer um conjunto de serviços verificados na sua arquitectura, encontram-se definidos na tabela 4.3. Para a representação da informação de cada um destes *URLs* são utilizadas as funções explicadas na secção 4.4.1. Assim sendo os *URLs* que se encontram relacionados com a activação de fluxos utilizam as funções descritas na secção 4.4.1.1. Por outro lado os *URLs* que pretendem efectuar a actualização de um fluxo usam as funções 4.4.1.2. Finalmente os *URLs* que eliminam fluxos na rede utilizam as funções apresentadas na secção 4.4.1.3.

URLs disponibilizados pelo módulo Activator		
URL	**VERB HTTP**	**Descrição**
/flow	GET	Devolve a lista dos grupos de fluxos possíveis
/flow/grouptag	GET	Devolve os *flows* existentes no grupo identificado pelo *group tag*
/flow/grouptag/flowid	GET	Devolve a informação do fluxo identificado pelo *flow ID*
/ARP/arpid	GET	Devolve a informação do fluxo identificado pelo *flow ID*
/flow/grouptag	POST	Cria um novo *flow* pertencente ao grupo identificado pelo *group tag*
/ARP	POST	Cria um novo *flow ARP*
/flow/grouptag/flowid	PUT	Actualiza a informação do fluxo identificado pelo *flow ID*
/flow/grouptag/flowid	DELETE	Elimina o fluxo identificado pelo *flow ID*
/ARP/arpid	DELETE	Elimina o fluxo identificado pelo *flow ID*

Tabela 4.3: URLs disponibilizados pelo módulo *Activator*

4.5 Conclusão

Neste capítulo procedeu-se à descrição da implementação da solução apresentada no capítulo 3 para construir o demonstrador de controlo de rede através de *Openflow*. Primeiro apresentaram-se as características físicas da *testbed* utilizada, e de seguida especificou-se o *software* e *hardware* característicos da *testbed*. Depois efectuou-se o estudo aprofundado sobre os módulos da solução que foram desenvolvidos no âmbito deste trabalho, apresentando a sua arquitectura interna, funcionalidades, algoritmos utilizados para verificar as especificações previstas no capítulo 3, e os recursos disponibilizados para efectuar a comunicação e gestão dos mesmos.

Capítulo 5

Testes e Análise dos Resultados

5.1 Introdução

Neste capítulo analisaremos a capacidade de resposta do demonstrador da solução apresentada, no capítulo 3, mediante a execução de um conjunto de experiências que têm como objectivo quantificar o desempenho dos módulos desenvolvidos assim como das suas funcionalidades. Para o módulo de monitorização foram propostas duas experiências, a primeira prende-se com a análise do tempo de resposta por parte do módulo à detecção de um novo *link* e da perda de um *link* na rede.

Para o módulo de activação propôs-se uma série de testes de desempenho que visam analisar o tempo de resposta do mesmo a uma série de pedidos de activação. Finalmente são desenvolvidos testes do sistema com um todo.

5.2 Análise ao módulo *Monitoring*

Na série de experiências que se seguem pretende-se realizar uma análise qualitativa do módulo *monitoring*, assim como analisar o desempenho do mesmo, nos cenários de perda de um *link* e a detecção de um novo *link* na rede.

5.2.1 *Testbed*

A *testbed* escolhida para a realização dos testes é a *testbed* apresentada no capítulo 4 deste trabalho, já que permite efectuar uma estimativa do tempo utilizado para perda de um *link* numa rede real *Openflow*.

5.2.2 Perda de *Link*

Esta experiência consiste em simular a perda de *link* de um dos nós da rede, e medir o tempo dispendido até esta ser detectada pelo módulo de *monitoring* e o tempo até esta informação ser enviada ao módulo imediatamente superior, que é o *Service Manager* de acordo com a arquitectura apresentada do gestor de rede no capítulo 3. Pretende-se que com este teste conseguir estimar o tempo médio utilizado pelo nosso módulo a detecção de problemas na rede.

5.2.2.1 Metodologia

Este teste será executado recorrendo as potencialidades da simulação, especificamente foi desenvolvido um *script*, na linguagem *Python*, que irá ser executado de forma local e que apresenta o fluxograma da figura 4.7:

A sequência de acções do *script* é:

- Inicialmente o *script* recolhe a topologia actual da rede.
- Seguidamente coloca a interface abaixo, momento em que é activo o nosso cronómetro.
- Neste momento o *script* entra em ciclo infinito. Volta a recolher a informação da topologia, com uma periodicidade de, aproximadamente, 20 Hz.
- Compara a topologia recolhida no ponto anterior com a recolhida no primeiro ponto. No caso de ser igual volta a comparar; em caso de serem diferentes dispara o alarme e verifica-se o tempo.

Para fins de teste o *link* que foi retirado da rede é o que efectua a ligação entre os *SW1* e *SW4* da figura 4.1. Notar que a topologia está a ser sempre comparada apartir do momento que entramos em ciclo infinito e que a periocidade de 20 Hz.

5.2.2.2 Resultados

Repetiu-se a experiência anterior 100 vezes, no *switch* 1 da figura 4.1, obtendo-se os resultados presentes na tabela 5.1 com um intervalo de confiança de 95 %

Resultados para o teste de Perda de *Link*		
Acção	**Tempo (s)**	**Intervalo de confiança(95 %)(s)**
Detectar	**0.019**	**± 0.005**
Envio	**0.026**	**± 0.007**
Tempo Total	**0.041**	**± 0.009**

Tabela 5.1: Tabela dos resultados obtidos para a simulação da perda de um *link* na rede

De acordo com os resultados presentes na tabela 5.1, pode-se concluir que o módulo *monitoring* demora 0.041 ± 0.09 segundos a detectar a perda de um *link* na rede.

5.2.3 Detecção de um novo *Link*

5.2.3.1 Metodologia

A metodologia a seguir por este teste é semelhante à utilizada no teste de perda de *link*, só que agora em vez de ser utilizado para detectar a perda de um *link*, é utilizado para a detectar a presença de um novo *link* na rede. Neste caso o *link* que serviu de teste foi o *link* que permite a comunicação entre o SW1 e SW4 da figura 4.1. O fluxograma que define o *script* é o 4.7 e apresenta a mesma ordem de sequência de acções.

5.2.3.2 Resultados

Repetiu-se a experiência anterior 100 vezes, no *switch* 1 da figura 4.1, obtendo-se os resultados presentes na tabela 5.2 com um intervalo de confiança de 95 %

De acordo com os resultados presentes na tabela 5.2, pode-se concluir que o módulo *monitoring* demora 2.81 ± 0.04 segundos a detectar a presença de um novo *link* na rede. Verifica-se que a detecção de um novo *link*

Resultados para o teste de Detecção de um novo *Link*		
Acção	**Tempo (s)**	**Intervalo de confiança(95 %)(s)**
Detectar	**2.78**	**± 0.040**
Envio	**0.022**	**± 0.004**
Tempo Total	**2.81**	**± 0.04**

Tabela 5.2: Tabela dos resultados obtidos para a simulação da obtenção de um novo *link* na rede

na rede, recorrendo a este processo de simulação, dispende mais tempo que a detecção da perda de um *link*, o que faz sentido visto que ao activar novamente a interface é preciso existir um processo de reconfiguração interna do *Personal Computer (PC)* , no qual está a correr o *openvswitch*. Depois desta configuração é notificada a interface do *OpenVswitch* e só finalmente o controlador.

5.2.3.3 Conclusão

A diferença de tempo verificada entre a detecção da perda de um *link* e a existência de um novo prende-se com o tempo gasto para a configuração interna da interface por parte da máquina em que se encontra localizada o *OpenVswitch*. Quando a interface é novamente ligada ao substrato, o computador que emula o *OpenVswitch* tem que realizar a reconfiguração da informação da interface, seguidamente notifica o *OpenVswitch* e finalmente este comunica ao *Floodlight* por meio das mensagens presentes na tabela 3.4. Na perda de um *link* só há o tempo gasto por notificar ao controlador a perda de um *link*, que como no nosso caso é local, apresenta um tempo de comunicação inferior. Assim sendo pode-se concluir que este processo de simulação faz sentido para a perda de um *link*, já para a notificação da presença de um novo *link* devia-se pensar noutro processo de detecção.

5.3 Análise ao módulo Activator

Na série de experiências que se seguem pretende-se realizar uma análise qualitativa da arquitectura do protótipo proposto no capítulo 3. Estas experiências visam analisar o desempenho da módulo *activator* no sistema.

No módulo *activator* optou-se por realizar um conjunto de testes que visam medir o desempenho das suas funcionalidades, mais concretamente a activação, actualização e eliminação das regras.

Estes testes serão divididos em duas partes, a primeira em que efectuaremos a activação de diversos serviços na *testbed* apresentada na secção 4.2 medindo os tempos de execução. Por outro lado, será analisado o comportamento do módulo para topologias maiores, isto é, topologias de 8, 16 e 32 nós.

5.3.1 *Testbed*

A primeira *testbed* a utilizar é a da figura 4.1, na qual se procedeu à activação de diversos serviços de conectividade.

Para topologias maiores, dado o elevado custo económico que representaria a construção de uma rede de maior dimensão, como a proposta na secção 4.2, optou-se por recorrer simulação da existências destas topologias, sendo necessário para tal desactivar a obtenção da topologia original da rede e inserir manualmente

a topologia desejada no módulo de *activator*. Desta forma conseguem-se aceitar os pedidos para este tipo de topologias. As figura 5.1, 5.2 e 5.3 ilustram a *testbed* de 8, 16 e 32 nós respectivamente.

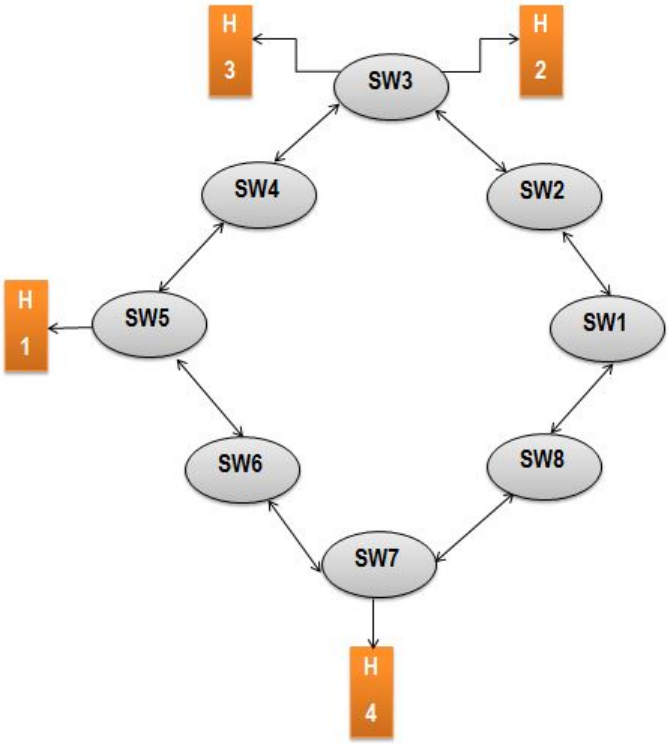

Figura 5.1: Topologia física de uma rede *Openflow* constituída por 8 nós

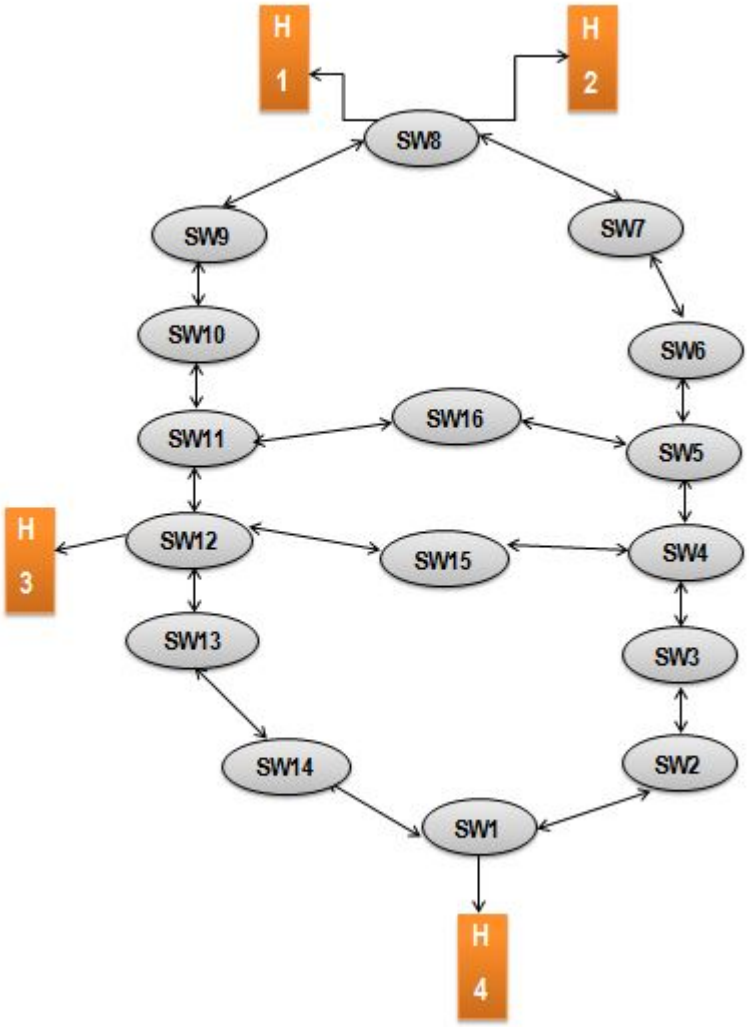

Figura 5.2: Topologia física de uma rede *Openflow* constituída por 16 nós

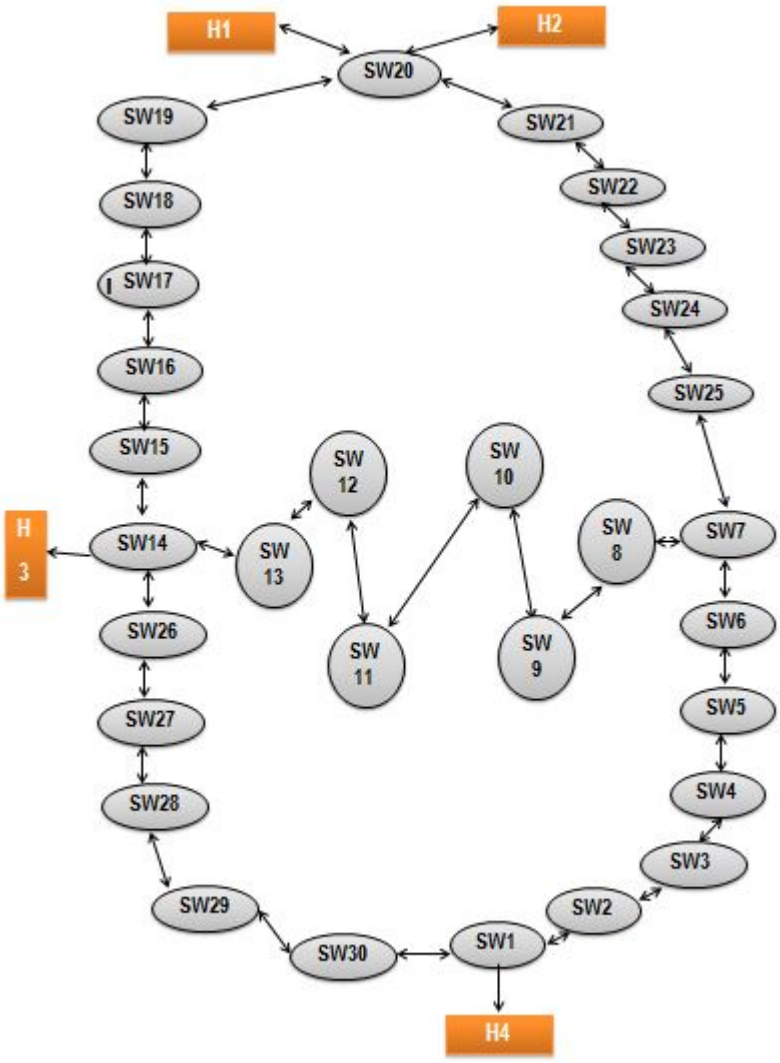

Figura 5.3: Topologia física de uma rede *Openflow* constituída por 32 nós

5.3.2 Metodologia

Para a realização destes testes considerou-se, dada a arquitectura interna e modo de funcionamento do módulo de activação, dividir o mesmo na medição de dois tempos, isto é, primeiro mediu-se o tempo que o módulo requer para realizar a tradução do pedido em regras para activar na rede e o tempo de activação na rede. Também é importante referir que este teste foi efectuado para diversos tipos de activação de serviços de conectividade tais como a activação de o serviço *TCP*, o serviço *UDP* e o serviço *ICMP*.

O teste consiste na chegada de pedidos simultâneos de activação de serviços de conectividade. Para tal desenvolveu-se um *script* cuja sequência de acções está apresentado no fluxograma da figura 4.9:

- Envio do pedido;
- Recepção do pedido, momento em que é activado o *Timer*;
- Decomposição e armazenamento do pedido em fluxos a activar na rede, medindo-se o tempo dispendido nesta acção;
- Activação do serviço e medição do tempo dispendido.

Note-se que para os testes de activação do serviço *ICMP* na rede considerou-se que na rede já estava activo o serviço *ARP* com os valores verificados na secção 5.3.3.

No caso da simulação realizada para topologias maiores, primeiramente fiz-se a alteração da topologia de forma estática no módulo *activator*, seguidamente a fim de garantir que todos os pedidos de activação apresentavam comunicação com o *Floodlight* efectou-se mais uma alteração estática nas funções com o objectivo das regras serem activas equitativamente nos *switches SW1* e *SW3* da figura 4.1. Desta forma mantinha-se a linha de pensamento realizada nos testes para a *testbed* original.

5.3.3 Teste de activação do serviço *ARP*

Realizou-se a experiência com uma metodologia semelhante a explicada na secção 5.3.2 para a activação do serviço *ARP* na rede para todos os elementos de rede que constituem as diversas topologias, isto é, para 4, 8, 16 e 32 nós. Para a topologia de 4 nós cada pedido de activação decompor-se-á na activação de uma regra. Para as topologias de 8, 16 e 32 nós utilizou-se uma lógica semelhante no momento de activação, só que agora cada pedido de activação decomponha-se em 2 regras para a topologia de 8 nós, de 4 regras para a topologia de 16 nós e de 8 regras para a topologia de 32 nós. Obtiveram-se os resultados representados na tabela para activação 5.3.

Resultados para a activação do serviço *ARP*		
Topologia	**Tempo (s)**	**Intervalo de confiança(95 %)(s)**
4 nós	**0.013**	± **0.002**
8 nós	**0.031**	± **0.005**
16 nós	**0.0775**	± **0.023**
32 nós	**0.1547**	± **0.030**

Tabela 5.3: Tabela dos resultados obtidos para a activação de um serviço *ARP* na rede

Estes resultados mostram que o tempo de activação de um serviço *ARP* apresenta um comportamento proporcional ao aumento da topologia, isto é, no caso da topologia de 4 nós o tempo médio de activação de cada uma das regras nos 4 *switches* é de 0.013 segundos. Ao aumentarmos a topologia o tempo começa

a aumentar proporcionalmente ao tamanho da topologia (aproximadamente), isto é, com a duplicação da topologia, verifica-se que o tempo de activação duplica aproximadamente. Os resultados encontram-se dentro do esperado, pois o tempo de activar 2 regras na rede e neste caso no mesmo *switch* é o dobro que o de activar uma regra, pois haveria de executar 2 decomposições e duas activações.

5.3.4 Teste de activação do serviço *TCP*

5.3.4.1 Topologia de 4 nós

Nesta secção realiza-se a experiência com a metodologia explicada na secção 5.3.2 para a activação de 50 serviços *TCP* diferentes para a rede representada da figura 4.1, em que 25 dos 50 serviços a inserir seguem o caminho SW4-SW1-SW2 e os outros percorrem o caminho SW4-SW3-SW2. Assim sendo, cada pedido irá ser decomposto em três regras, uma em cada *switch*, a activar na rede.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.4 e 5.5. A activação de um serviço *TCP* no substrato de 4 nós, que se decompõe em 1 fluxo que por sua vez é traduzido em 3 regras, demora 41 $\pm$ 0.005 ms a ficar activo na rede como se pode observar na figura 5.4. Outra conclusão que se pode tirar da figura 5.4 é que o processo de activação de fluxos na rede apresenta um comportamento limite, isto é, com o aumento do número de fluxos a ser activos na rede este tende para um valor máximo.

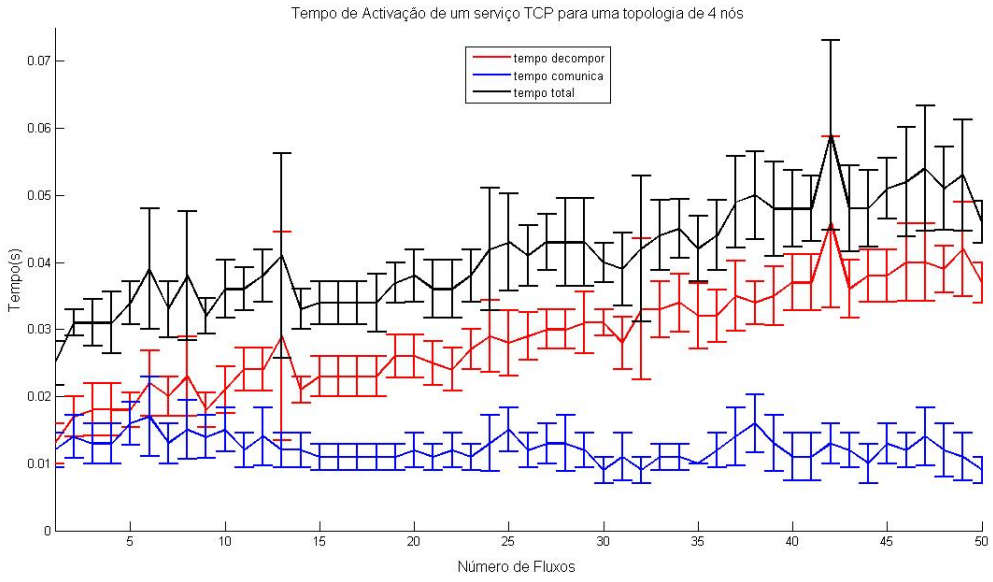

Figura 5.4: Tempo para a Activação de um fluxo *TCP* no substrato de 4 nós

Por outro lado, após o tratamento dos resultados obtidos da figura 5.4, obtêm-se os resultados apresentados no gráfico da figura 5.5, na qual se observa que a activação de 5 fluxos *TCP* demora aproximadamente 0.152 $\pm$ 0.018 segundos e a activação de 50 fluxos *TCP* demora 2.05 segundos. Assim podemos concluir que com o aumento do número de fluxos a activar na rede verifica-se um comportamento aproximadamente linear. Também se verifica que o tempo de decomposição e comunicação aumentam; contudo o tempo de decomposição aumenta mais rapidamente que o processo de comunicação, concluindo-se assim que a componente de decomposição é a que mais contribui para o tempo total de activação do fluxo na rede.

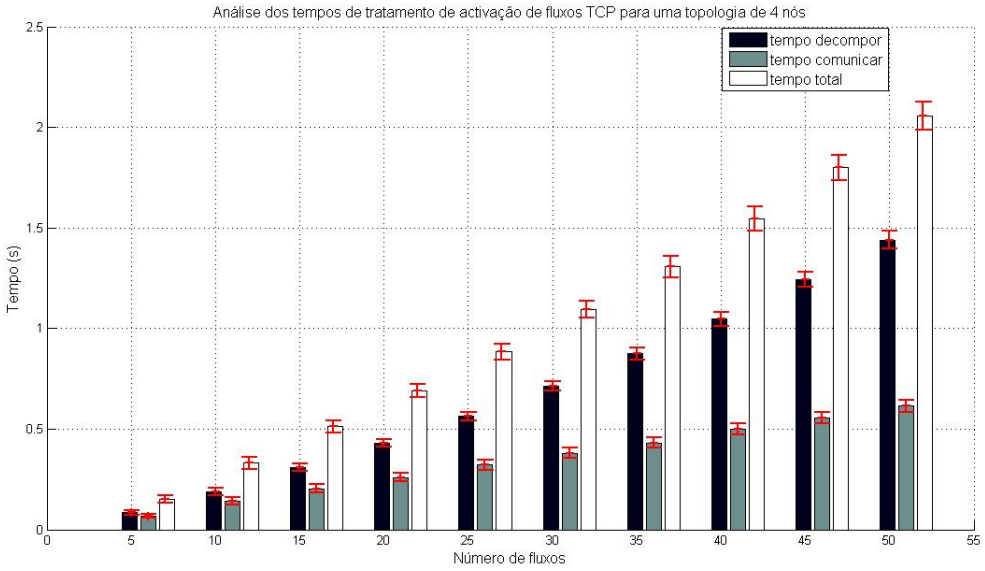

Figura 5.5: Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 4 nós

---

5.3.4.2 Topologia de 8 nós

Nesta secção realiza-se a experiência com a metodologia explicada na secção 5.3.2 para a activação de 50 serviços *TCP*, para a rede representada da figura 5.1, em que 25 dos 50 serviços a inserir seguem o caminho SW7-SW8-SW1-SW2-SW3 e as outras percorriam o caminho SW7-SW6-SW5-SW4-SW3. Assim sendo, cada pedido irá ser decomposto em 5 regras a activar na rede.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.6 e 5.7.

A activação de um serviço *TCP* no substrato de 8 nós decompõe-se em 1 fluxo que por sua vez é traduzido em 5 regras, demora 0.068 ± 0.005 segundos a ficar activo na rede como se pode observar na figura 5.6. Outra conclusão que se pode tirar da figura 5.6 é que o processo de activação de fluxos na rede apresenta um comportamento limite, isto é, com o aumento do número de fluxos a ser activos na rede este tende para um valor máximo.

Da análise dos resultados obtidos no gráfico da figura 5.7, observa-se que a activação de 5 fluxos *TCP* demora aproximadamente 0.295 ± 0.045 segundos e a activação de 50 fluxos *TCP* demora 3.39 ± 0.074 segundos. Verifica-se em relação à topologia anterior um aumento do tempo de activação dos fluxos, este não duplica como se estaria à espera devido ao fluxo ser decomposto em 5 regras e não em 6 regras. Também continua a se verificar o comportamento aproximadamente linear dos tempos de decomposição e comunicação na rede, sendo a componente de decomposição a que mais contribui para o tempo total de activação.

5.3.4.3 Topologia de 16 nós

Esta secção contém a experiência que realiza a activação de 50 serviços *TCP* diferentes para uma rede constituída por 16 nós, ver figura 5.2, obrigando a que cada um dos pedidos dos fluxos seja decomposto no dobro das regras pressupostas no teste anterior.

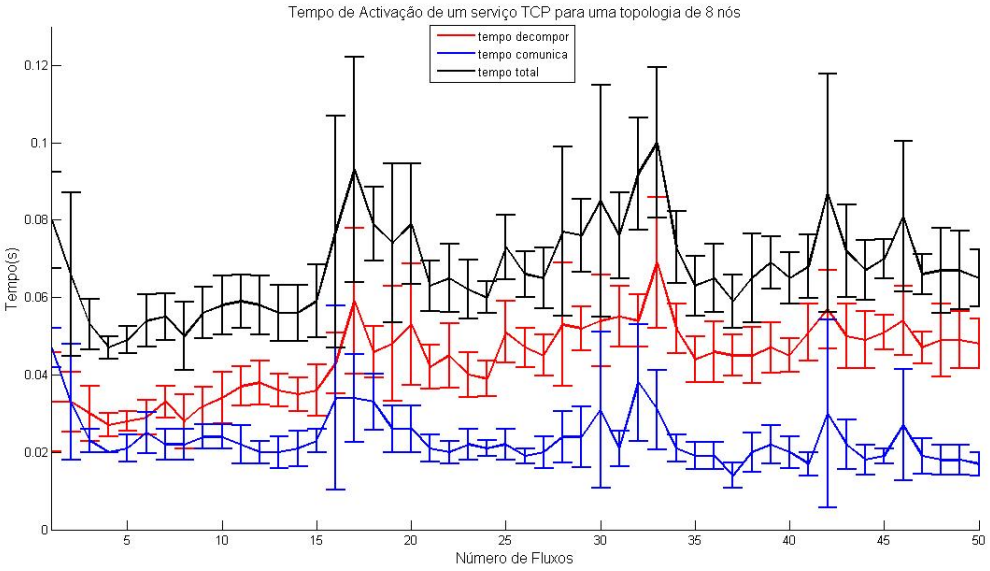

Figura 5.6: Tempo para a Activação de um fluxo *TCP* no substrato de 8 nós

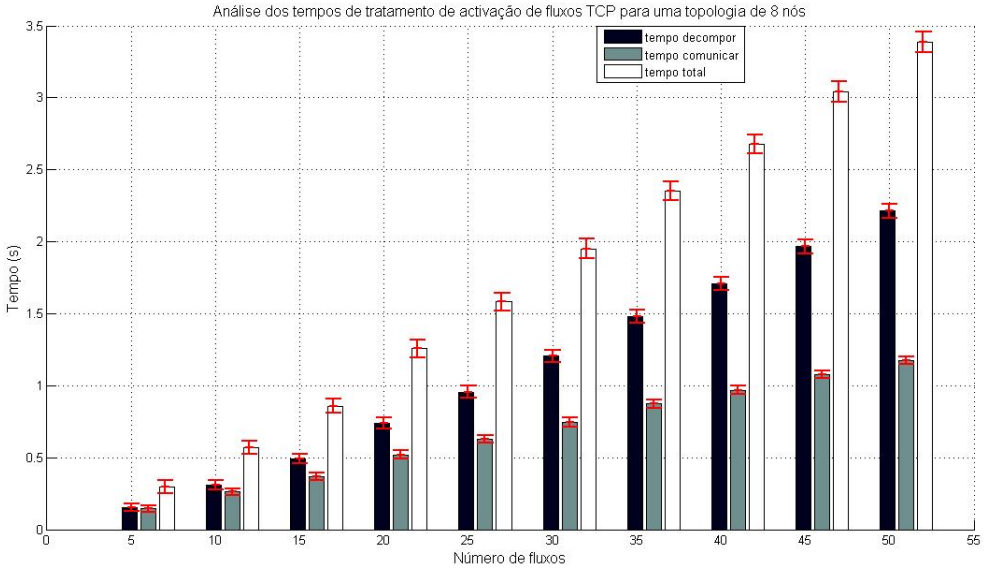

Figura 5.7: Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 8 nós

---

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.8 e 5.9.

A activação média de um destes pedidos demora 0.163 ± 0.022 segundos a ficar activo na rede. Como era expectavel o tempo de activação duplico em relação a experiência anterior, onde um fluxo dava origem a 5

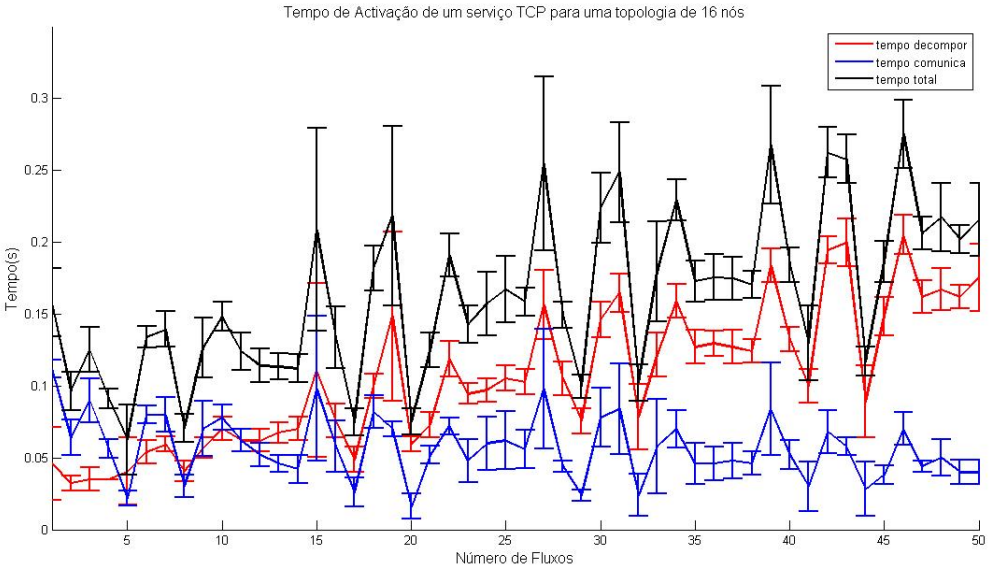

Figura 5.8: Tempo para a Activação de um fluxo *TCP* no substrato de 16 nós

---

regras, neste caso um fluxo é decomposto em 10 regras.

Do gráfico da figura 5.8 pode-se concluir que o processo de activação de fluxos na rede apresenta um comportamento limite. Embora no início se verifique que a comunicação apresenta maior influência no tempo total de activação dos fluxos, esta anomalia pode estar relacionada com a utilização do *Central Processing Unit (CPU)* nos instantes em que foram efectuados os testes, prejudicando os resultados iniciais. Contudo, depois verifica-se que é a componente de decomposição que maior contribuição tem no tempo total de activação.

A activação de 5 fluxos *TCP* demora aproximadamente 0.533 ± 0.072 segundos e a activação de 50 fluxos *TCP* demora 8.15 ± 0.087 segundos. Verifica-se que com a duplicação do número de regras em que são traduzidos os fluxos, o tempo de decomposição e de comunicação também aumentam. Embora no início a componente de comunicação seja ligeramente superior à de decomposição devido às razões explicadas anteriormente, a componente que mais contribui para o tempo total de activação dos fluxos é o tempo de decompor os fluxos em regras.

5.3.4.4 Topologia de 32 nós

Para a activação de 50 serviços *TCP* diferentes para uma rede constituída por 32 nós, partir-se-á das topologias utilizadas na *testbed* de 32 nós, ver figura 5.3, obrigando a que cada um dos pedidos dos fluxos seja decomposto no dobro das regras pressupostas no teste anterior.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.10 e 5.11.

A activação média de um destes pedidos demora 0.331 ± 0.037 ms a ficar activo na rede. Do gráfico da figura 5.10 pode-se concluir que o processo de activação de fluxos na rede apresenta um comportamento limite, ou seja, com o aumento do número de fluxos a ser activos na rede este tende para um valor máximo. Como era de esperar, o tempo de activação duplicou em relação ao tempo de activação da topologia de 16 nós e ficou aproximadamente 8 vezes maior que o tempo verificado para a topologia real de 4 nós.

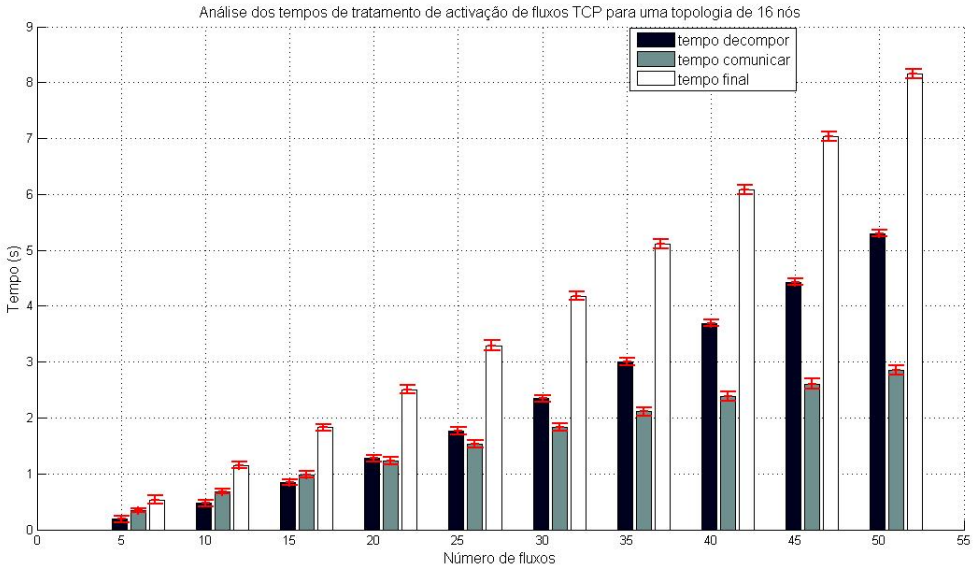

Figura 5.9: Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 16 nós

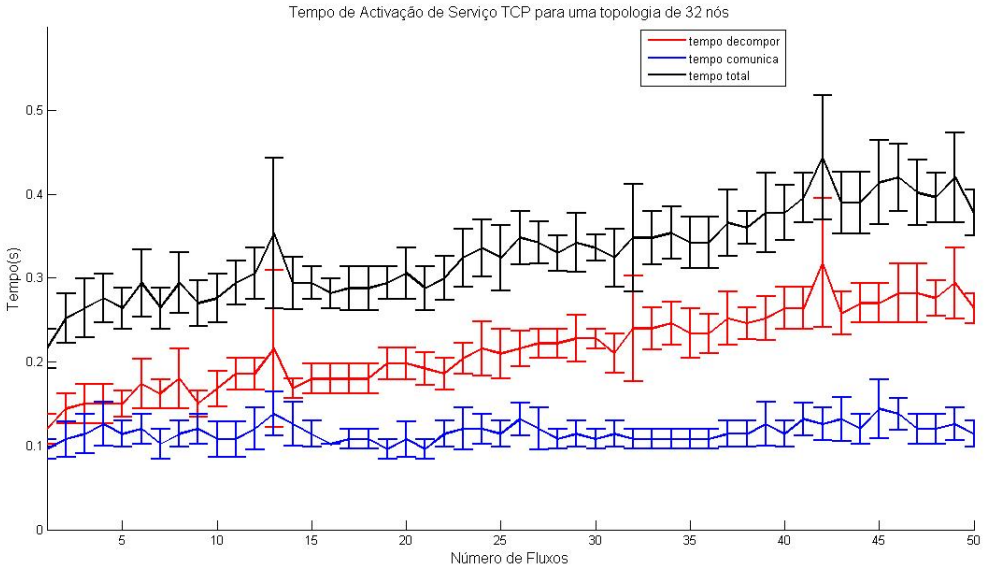

Figura 5.10: Tempo para a Activação de um fluxo *TCP* no substrato de 32 nós

---

Depois da análise e tratamento dos resultados obtidos no gráfico da figura 5.11 observa-se que a activação de 5 fluxos *TCP* demora aproximadamente 1.27 $\pm$ 0.152 segundos, e a activação de 50 fluxos *TCP* demora 16.53 $\pm$ 0.414 segundos. Relativamente à experiência anterior verifica-se que o tempo de decomposição e

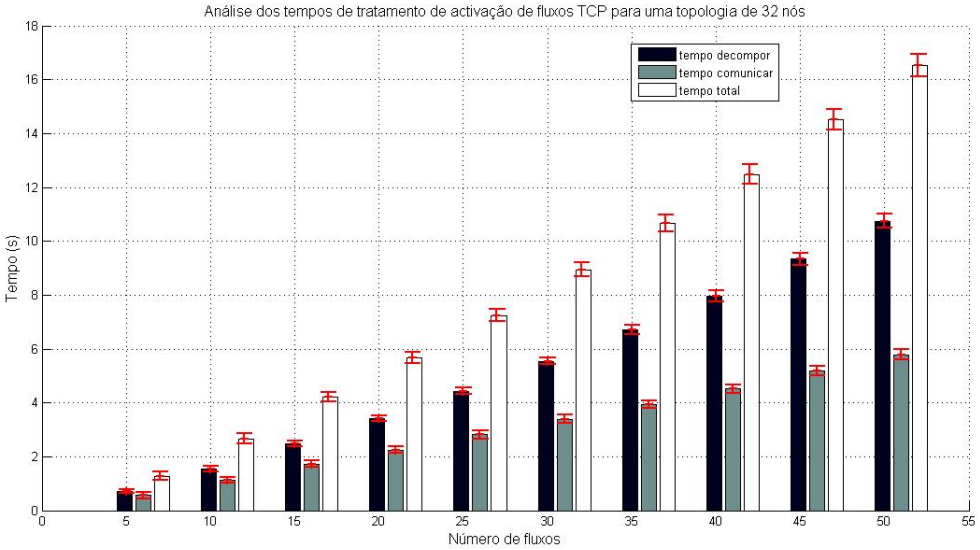

Figura 5.11: Variação do tempo de activação com o aumento do número de fluxos *TCP* no substrato de 32 nós

---

comunicação duplicam com a duplicação do número de regras a activar por *switch* na rede, como era esperado, mantendo assim um comportamento aproximadamente linear.

5.3.4.5 Conclusão

Como conclusão geral dos testes sobre a activação de um serviço *TCP* pode-se dizer que se verifica uma relação entre o aumento da topologia e o tempo de activação, verificando os resultados esperados, pois com a duplicação da topologia nos diversos testes duplica-se o número de fluxos a activar na rede. Também se verifica que o tempo de decomposição aumenta com o número de fluxos a activar, novamente pela razão anteriormente explicada. Enquanto que o tempo de comunicação com o aumento do número de fluxos apresenta menos variações, podendo ser uma das razões a ligação *HTTP* que tende a apresentar melhores resultados para os últimas partes da informação. Assim, podemos concluir que a fase de decomposição é aquela que tem maior influência no tempo total de activação.

5.3.5 Teste de activação do serviço *UDP*

Nesta secção realiza-se um estudo semelhante ao efectuado na secção 5.3.4 esperando-se, dada as características semelhantes entre a activação de uma regra *TCP* e uma regra *UDP*, que os resultados e conclusões a retirar sejam semelhantes aos da secção anterior.

5.3.5.1 Topologia de 4 nós

Nesta secção realiza-se a experiência de activação de 50 serviços *UDP* diferentes para a rede representada da figura 4.1, em que 25 dos 50 serviços a inserir seguem o caminho SW4-SW1-SW2 e os outros percorrem o

caminho SW4-SW3-SW2. Assim sendo, cada pedido irá ser decomposto em três regras, uma em cada *switch*, a activar na rede.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.12 e 5.13.

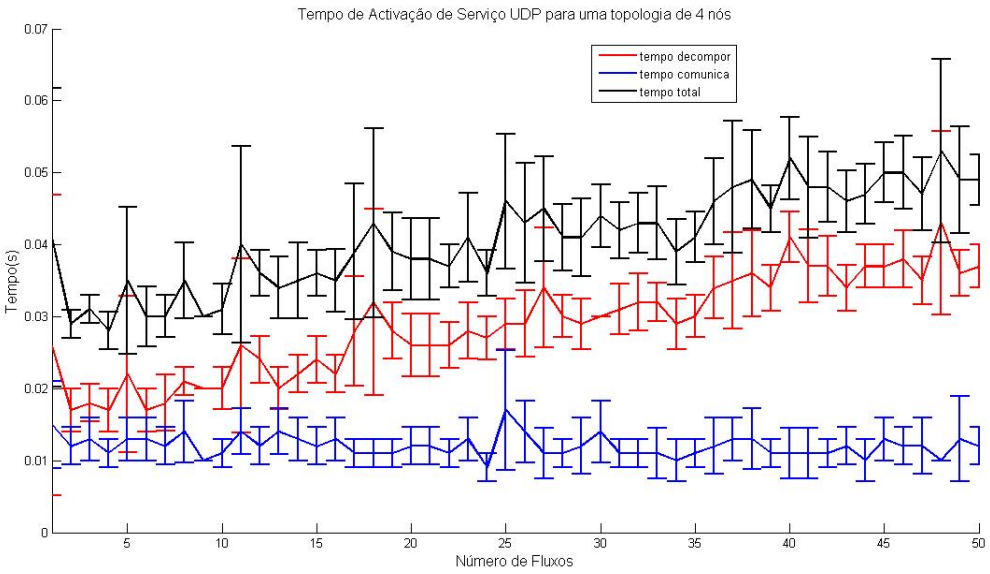

Figura 5.12: Tempo para a Activação de um fluxo *UDP* no substrato de 4 nós

À semelhança dos resultados verificados para a activação de um fluxo *TCP*, a activação de um serviço *UDP* no substrato de 4 nós, que se decompõe em 1 fluxo que por sua vez é traduzido em 3 regras, demora 41 $\pm$ 0.005 ms a ficar activo na rede como se pode observar na figura 5.12. Mantém um comportamento limite, isto é, com o aumento do número de fluxos a ser activos na rede este tende para um valor máximo, verificando-se assim um comportamento semelhante ao serviço *TCP* o que faz sentido pois o serviço *UDP* e *TCP* são decompostos no mesmo número de regras. No gráfico da figura 5.13 observa-se que a activação de 5 fluxos *UDP* demora aproximadamente 0.164 $\pm$ 0.031 segundos e a activação de 50 fluxos *TCP* demora 2.05 segundos, valores muito próximos dos apresentados pelo fluxo *TCP*. Também se observa um comportamento aproximadamente linear por parte dos tempo de comunicação e activação com o aumento do número de fluxos a activar.

5.3.5.2 Topologia de 8 nós

Nesta secção realiza-se a experiência de activação de 50 serviços *UDP*, para a rede representada da figura 5.1, em que 25 dos 50 serviços a inserir seguem o caminho SW7-SW8-SW1-SW2-SW3 e as outras percorriam o caminho SW7-SW6-SW5-SW4-SW3. Assim sendo, cada pedido irá ser decomposto em 5 regras a activar na rede.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.14 e 5.15.

A activação de um serviço *UDP* no substrato de 8 nós, que se decompõe em 1 fluxo que por sua vez é traduzido em 5 regras, demora 0.067 $\pm$ 0.012 segundos a ficar activo na rede como se pode observar na figura 5.14. No gráfico da figura 5.15 observa-se que a activação de 5 fluxos *UDP* demora aproximadamente 0.313 $\pm$ 0.073 segundos e a activação de 50 fluxos *UDP* demora 3.35 $\pm$ 0.103 segundos. Novamente, e à semelhança do

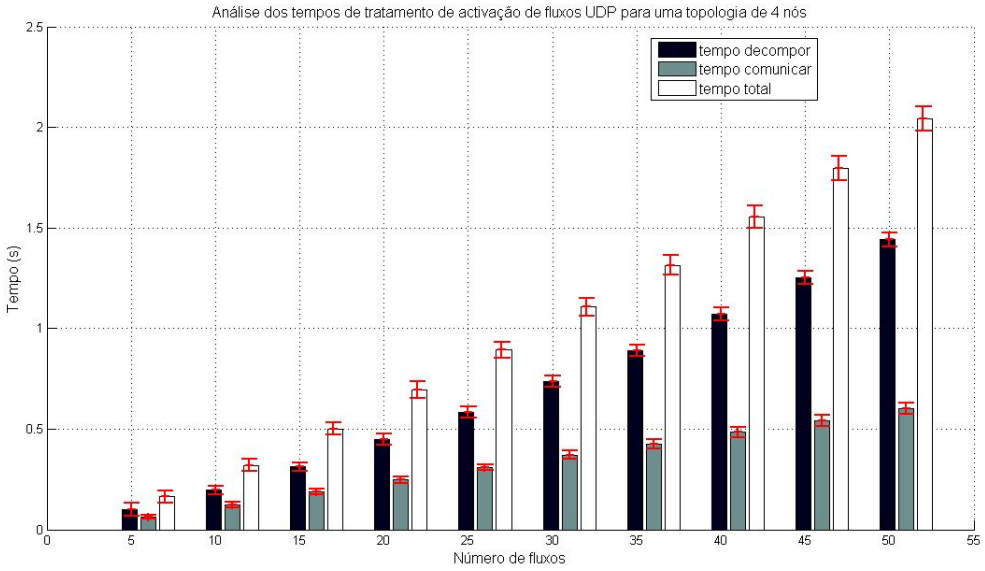

Figura 5.13: Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 4 nós

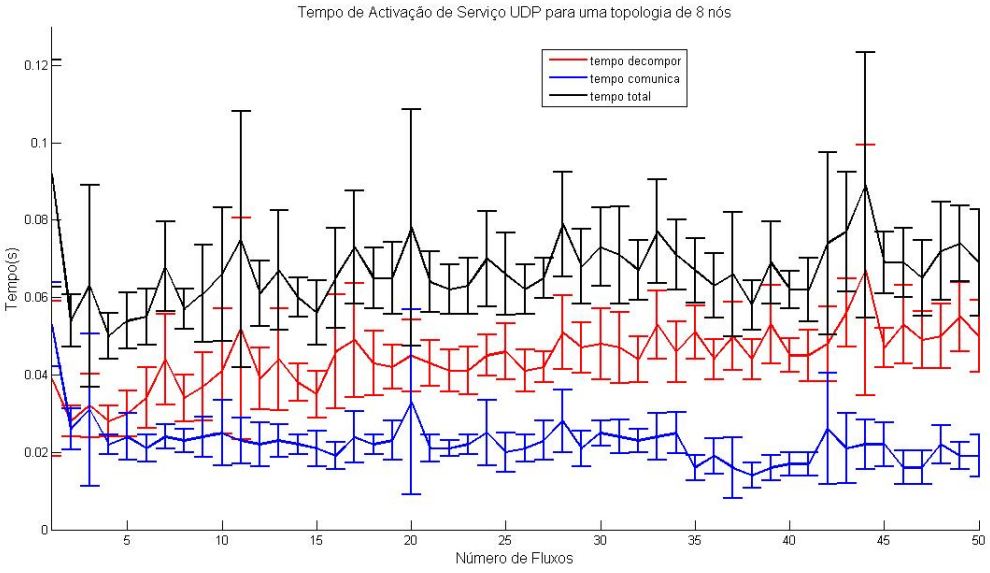

Figura 5.14: Tempo para a Activação de um fluxo *UDP* no substrato de 8 nós

---

que aconteceu na activação do fluxo *TCP*, verifica-se que com a duplicação dos dispositivos de rede o tempo dispendido pela módulo também dúplica. Neste caso não se verifica essa proporcionalidade directa porque activamos só 5 regras e não as 6 que verificariam este comportamento.

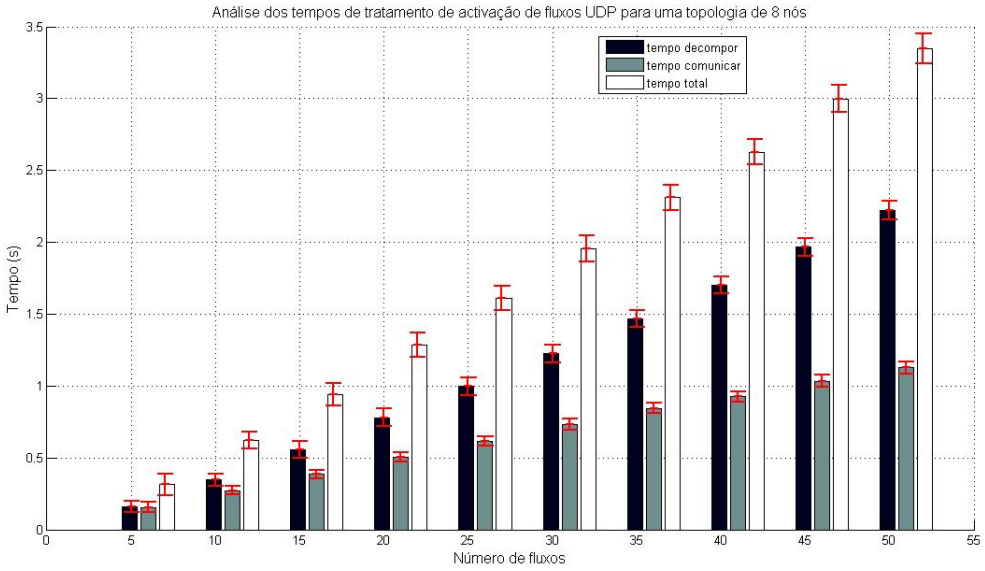

Figura 5.15: Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 8 nós

---

5.3.5.3 Topologia de 16 nós

Esta secção descreve a experiência que realiza a activação de 50 serviços *UDP* diferentes para uma rede constituída por 16 nós, ver figura 5.2, obrigando a que cada um dos pedidos dos fluxos seja decomposto no dobro das regras pressupostas no teste anterior.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.16 e 5.17.

A activação de um serviço *UDP* no substrato de 16 nós, demora 0.163 ± 0.022 segundos a ficar activo na rede como se pode observar na figura 5.16. Da análise dos resultados obtidos no gráfico da figura 5.17 observa-se que a activação de 5 fluxos *UDP* demora aproximadamente 0.560 ± 0.102 segundos e a activação de 50 fluxos *UDP* demora 8.23 ± 0.118 segundos. Desta forma, verificam-se assim os resultados esperados deste teste. Embora os resultados sejam semelhantes aos apresentados pela activação de um fluxo *TCP*, verifica-se certa instabilidade do nosso módulo, mais concretamente na parte da comunicação, onde se verifica na figura 5.16 que nalguns instante o tempo de comunicação ultrapassa o tempo de decomposição. Em princípio, este problema estará relacionado com a utilização interna do *CPU*.

5.3.5.4 Topologia de 32 nós

Para a activação de 50 serviços *UDP* diferentes para uma rede constituída por 32 nós, partir-se-á das topologias utilizadas na *testbed* de 32 nós, , ver figura 5.3, obrigando a que cada um dos pedidos dos fluxos seja decomposto no dobro das regras pressupostas no teste anterior.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.18 5.19.

A activação de um serviço *UDP* no substrato de 32 nós demora 0.329 ± 0.037 segundos a ficar activo na rede como se pode observar na figura 5.18. Da análise dos resultados obtidos no gráfico da figura 5.19 observa-se que a activação de 5 fluxos *UDP* demora aproximadamente 1.37 ± 0.102 segundos, e a activação de 50

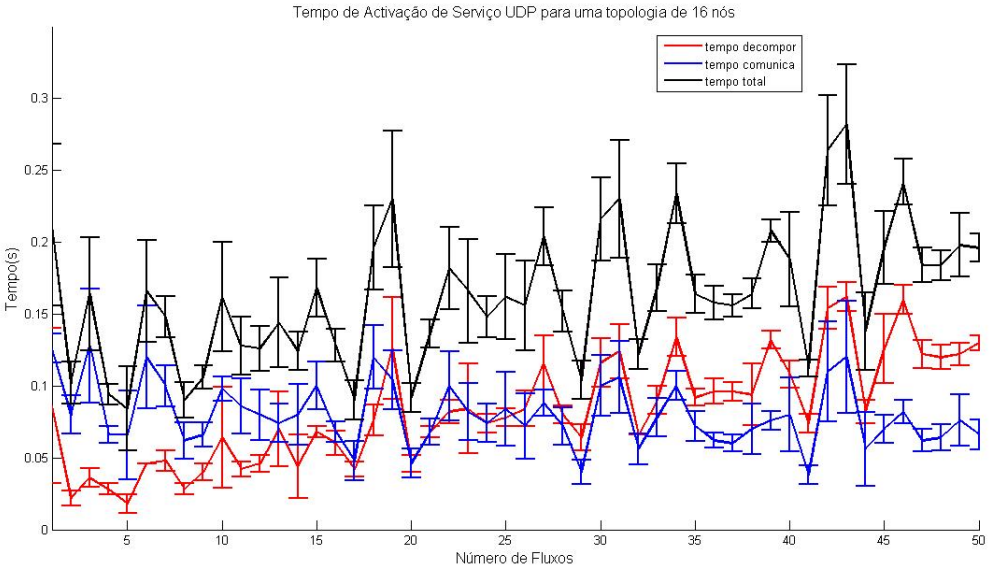

Figura 5.16: Tempo para a Activação de um fluxo *UDP* no substrato de 16 nós

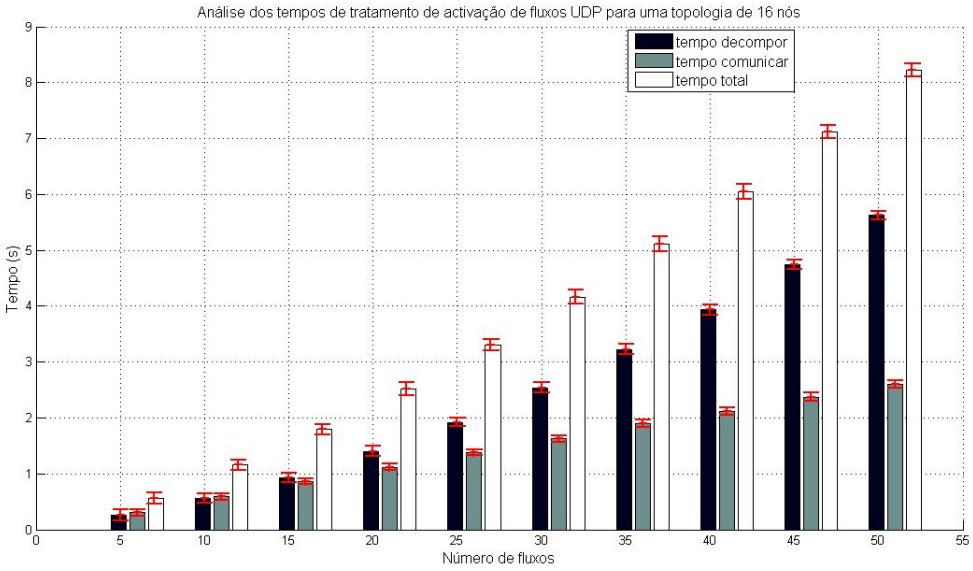

Figura 5.17: Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 16 nós

---

fluxos *UDP* demora 16.45 $\pm$ 0.366 segundos. Com o aumento do número de fluxos a activar na rede verifica-se que o tempo de decomposição e comunicação aumentam. Contudo, o tempo de decomposição aumenta mais rapidamente que o processo de comunicação, concluindo-se assim que a componente de decomposição é a que

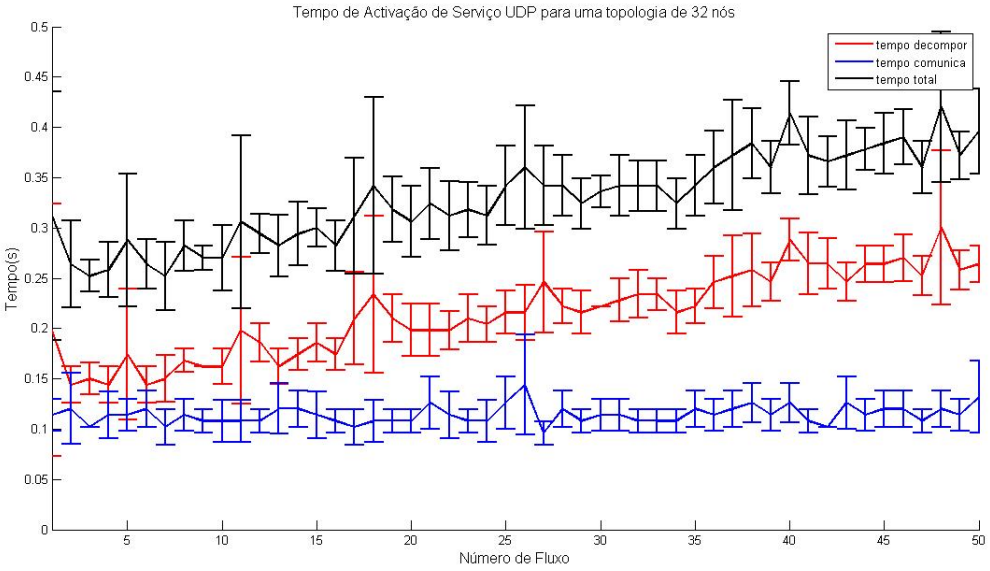

Figura 5.18: Tempo para a Activação de um fluxo *UDP* no substrato de 32 nós

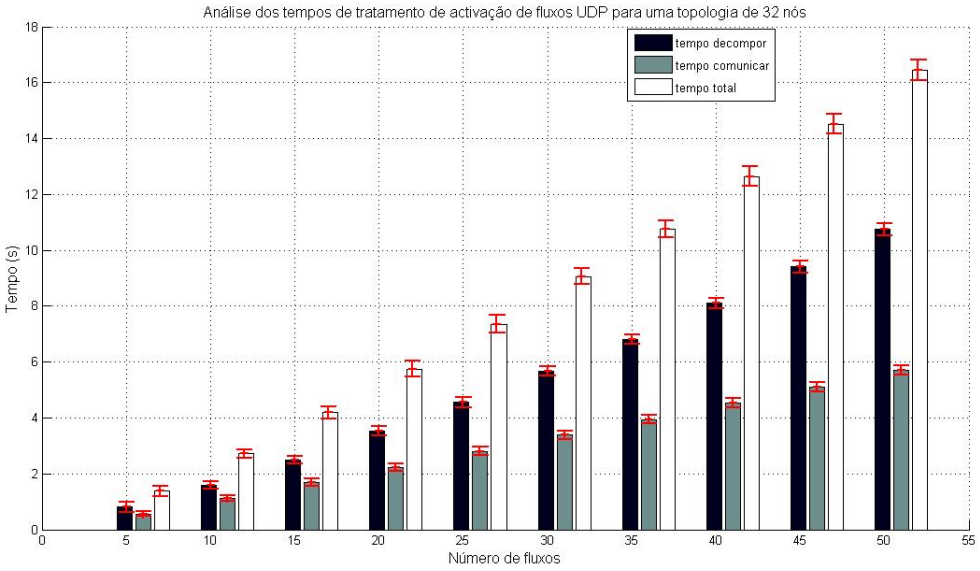

Figura 5.19: Variação do tempo de activação com o aumento do número de fluxos *UDP* no substrato de 32 nós

---

mais contribui para o tempo total de activação do fluxo na rede.

5.3.5.5 Conclusão

Como conclusão geral dos testes sobre a activação de um serviço *UDP* pode-se dizer que verifica um comportamenteo semelhante ao verificado pela activação de um serviço *TCP*, o que faz sentido e está dentro dos resultados esperados, pois o único parâmetro que varia na activação de uma regra destes serviços é a indicação do número do protocolo.

5.3.6 Teste de activação do serviço *ICMP*

Nesta secção realiza-se um estudo semelhante ao efectuado nas secções 5.3.4 e 5.3.5. Na activação de um serviço *ICMP*, para seu correcto funcionamento, deve ser indicado o caminho de ida e o caminho de volta, assim como activar o serviço *ARP*. Por tal motivo nas seguintes experiências considerou-se que o serviço *ARP* já se encontrava activo na rede, e cada pedido de activação de um serviço *ICMP* implicava a activação do fluxo de ida e o fluxo de volta, esperando-se assim que os tempos fossem maiores entre a activação de um pedido *UDP* ou *TCP*.

5.3.6.1 Topologia de 4 nós

Nesta secção realiza-se a experiência de activação de 50 serviços *ICMP* diferentes para a rede representada da figura 4.1. Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.20 e 5.21.

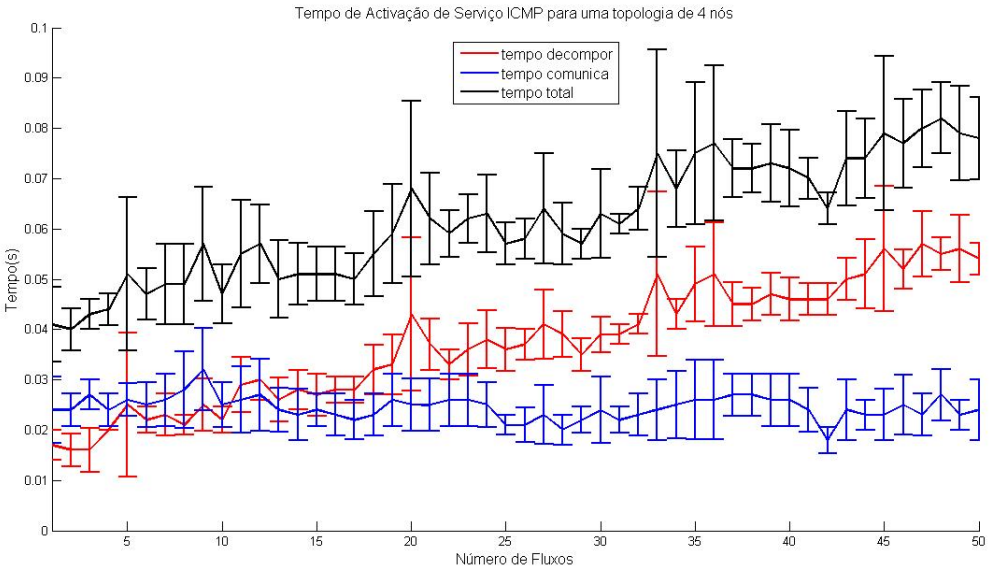

Figura 5.20: Tempo para a Activação de um fluxo *ICMP* no substrato de 4 nós

Pela observação dos resultados, verifica-se um comportamento limite a tender para um valor máximo. Como era de esperar, o tempo de activação dos fluxos aumentou, mas não com a proporcionalidade desejada. Esta diferença podera dever-se ao facto de não se considerar os tempos de activação do serviço *ARP*, e segundo por se verificar certa instabilidade na comunicação nos instantes iniciais.

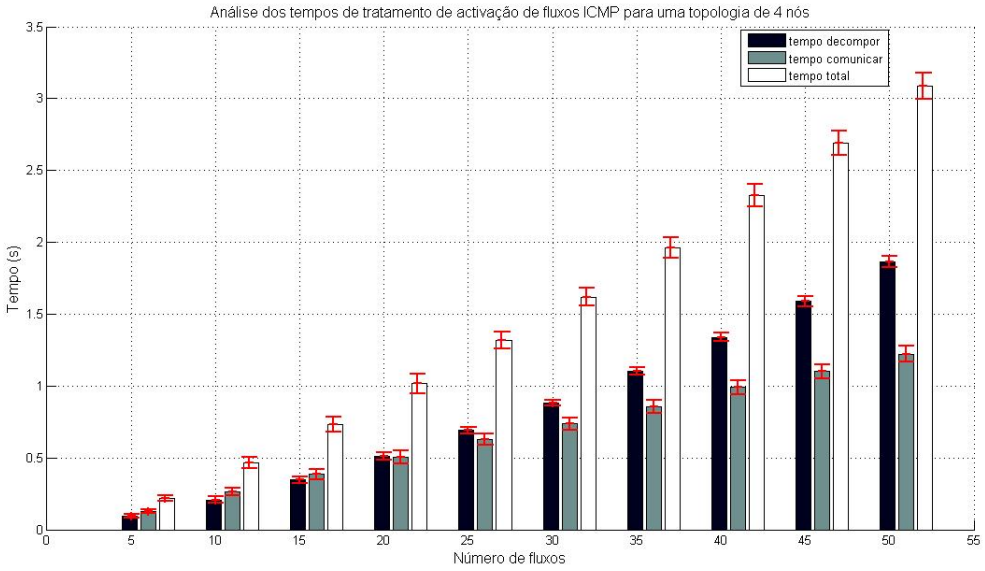

Figura 5.21: Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 4 nós

---

5.3.6.2 Topologia de 8 nós

Nesta secção realiza-se a experiência de activação de 50 serviços *ICMP*, para a rede representada da figura 5.1. Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.22 e 5.23.

A activação de um serviço *ICMP* no substrato de 8 nós, que se decompõe em 1 fluxo que por sua vez é traduzido em 10 regras, demora 0.100 ± 0.013 segundos a ficar activo na rede como se pode observar na figura 5.22. No gráfico da figura 5.23 observa-se que a activação de 5 fluxos *ICMP* demora aproximadamente 0.434 ± 0.052 segundos e a activação de 50 fluxos *ICMP* demora 4.96 ± 0.115 segundos. Os resultados apresentam-se dentro do expectável, visto que não há uma duplicação directa do número de regras de acordo com a experiência passada, e verifica-se um aumento do tempo de activação comparativamente com os serviços *UDP* e *TCP*.

5.3.6.3 Topologia de 16 nós

Esta secção contém a experiência de activação de 50 serviços *ICMP* diferentes para uma rede constituída por 16 nós, ver figura 5.2, obrigando a que cada um dos pedidos dos fluxos seja decomposto no dobro das regras pressupostas no teste anterior.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.24 e 5.25.

Verifica-se, como esperado, uma duplicação do tempo de activação com a duplicação da topologia. Em relação à experiência passada verifica-se uma duplicação dos tempos de activação, como esperado. Por outro lado, o tempo de activação, comparativamente aos serviços *UDP e TCP* aumentam. Contudo, verifica-se uma elevada variação e instabilidade no processo de comunicação e de decomposição das regras, que deve estar relacionado com a utilização do *CPU* no instante em que se efectou a experiência.

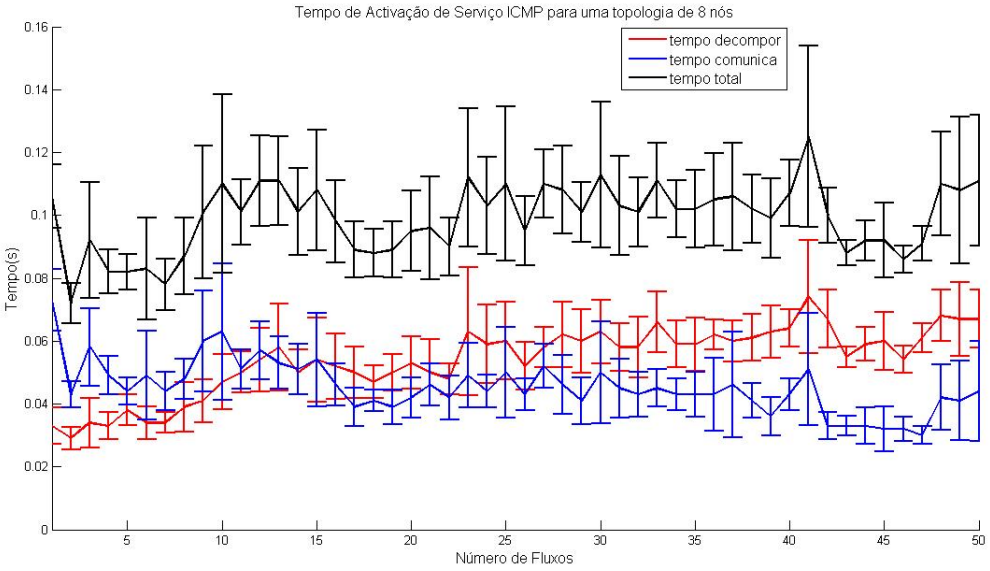

Figura 5.22: Tempo para a Activação de um fluxo *ICMP* no substrato de 8 nós

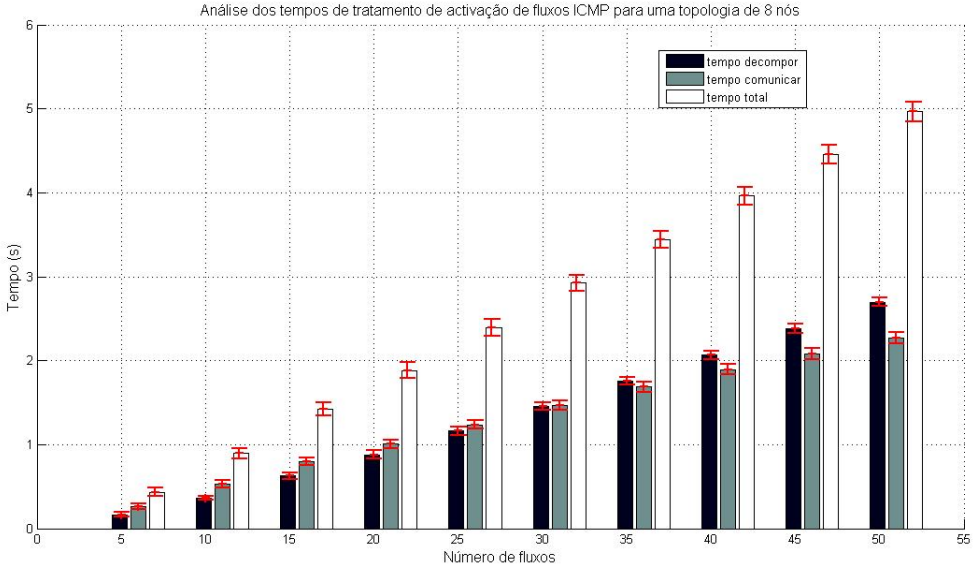

Figura 5.23: Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 8 nós

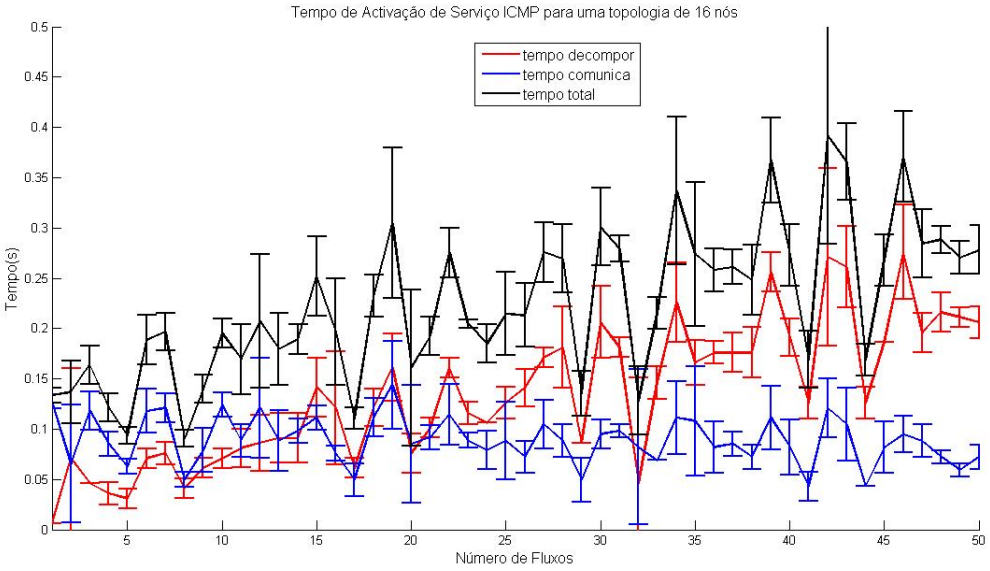

Figura 5.24: Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 16 nós

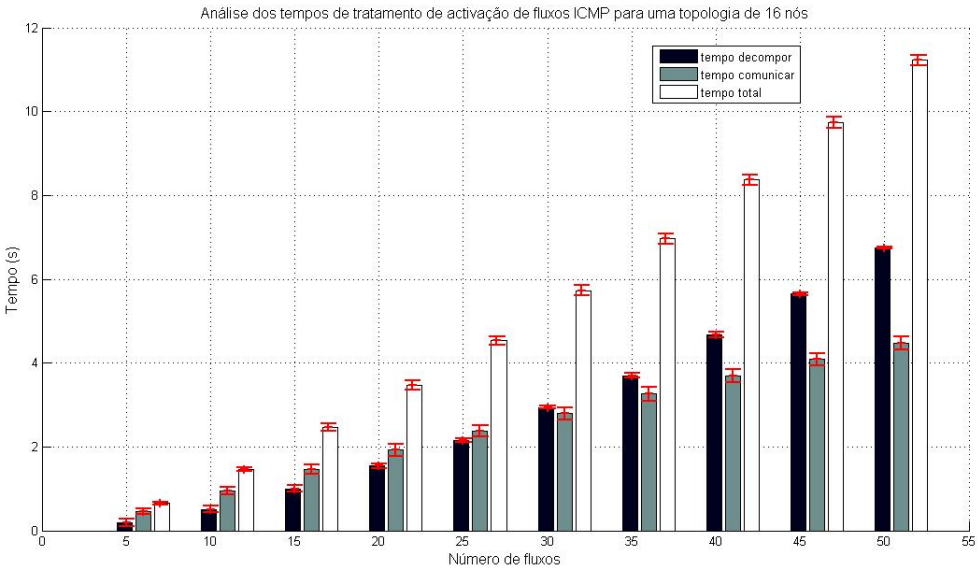

Figura 5.25: Tempo para a Activação de um fluxo *ICMP* no substrato de 16 nós

5.3.6.4 Topologia de 32 nós

Para a activação de 50 serviços *ICMP* diferentes para uma rede constituída por 32 nós, partir-se-á das topologias utilizadas na *testbed* de 32 nós, , ver figura 5.3, obrigando a que cada um dos pedidos dos fluxos

seja decomposto no dobro das regras pressupostas no teste anterior.

Obtiveram-se os seguintes resultados, que são representados nos gráficos das figuras 5.26 e 5.27.

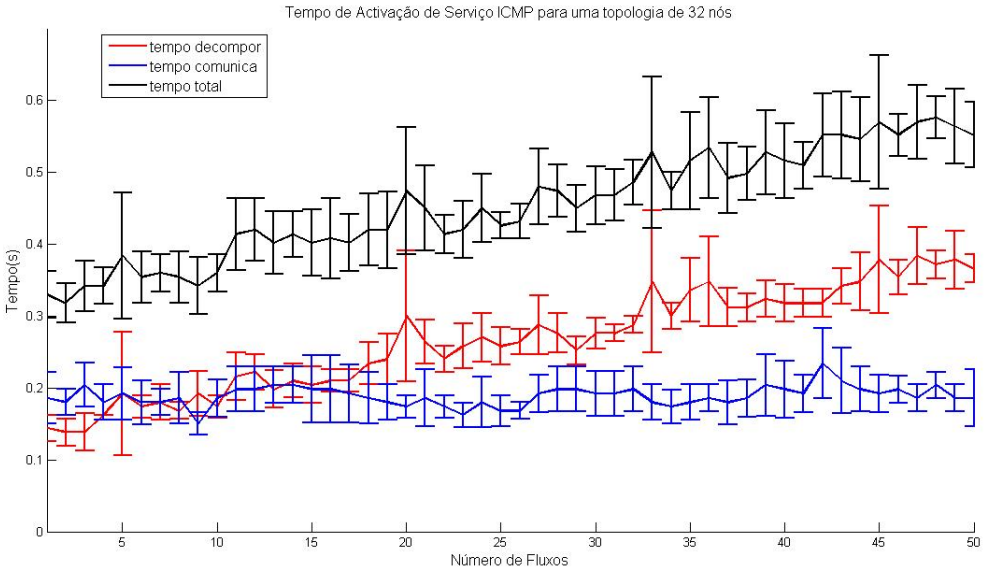

Figura 5.26: Tempo para a Activação de um fluxo *ICMP* no substrato de 32 nós

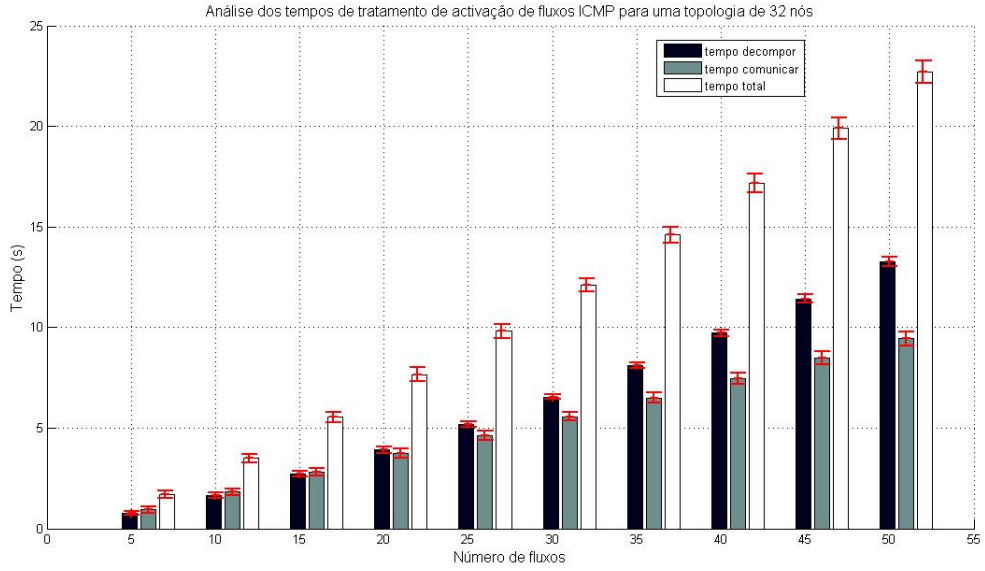

Figura 5.27: Variação do tempo de activação com o aumento do número de fluxos *ICMP* no substrato de 32 nós

Verifica-se assim uma duplicação relativamente aos valores obtidos na experiência anterior e um aumento do tempo de activação comparativamente aos serviços *TCP* e *UDP*.

5.3.6.5 Conclusão

Como conclusão geral dos testes sobre a activação de um serviço *ICMP* verifica-se uma elevada variação no tempo de comunicação e de decomposição, estando estes muito próximos, existindo assim certa interferência entre as componentes, o que pode contribuir com o fact de não verificar o resultado esperado em relação aos serviços *TCP* e *UDP*. O facto de considerar que o serviço *ARP* já se encontrava activo também contribui para as diferenças de valores.

5.3.7 Resultados Sumários

Esta secção apresenta uma comparação dos resultados nos diferentes serviços e topologias.

Resultados da activação de 50 fluxos para os diversos serviços e topologias			
Topologia	**Serviço**	**Tempo (s)**	**Intervalo de confiança(95 %)(s)**
4 nós	*UDP*	**0.041**	± **0.006**
	TCP	**0.041**	± **0.005**
	ICMP	**0.062**	± **0.008**
8 nós	*UDP*	**0.067**	± **0.012**
	TCP	**0.068**	± **0.010**
	ICMP	**0.100**	± **0.013**
16 nós	*UDP*	**0.163**	± **0.022**
	TCP	**0.163**	± **0.020**
	ICMP	**0.224**	± **0.031**
32 nós	*UDP*	**0.329**	± **0.037**
	TCP	**0.331**	± **0.034**
	ICMP	**0.454**	± **0.046**

Tabela 5.4: Tabela dos resultados obtidos para a simulação da perda de um *link* na rede

A partir dos resultados presentes na tabela 5.4, pode-se concluir que a activação de um serviço *UDP* e um serviço *TCP* demoram, aproximadamente, o mesmo tempo para ficarem activos na rede, enquanto que o serviço *ICMP* dispende mais tempo a ficar activo na rede, estaria a espera que fosse o dobro do tempo, devido ao facto de a sua activação implica que um mesmo fluxo seja decomposto no dobro do número de regras, pois na implementação do módulo optou-se por realizar que o *ICMP* request e *ICMP* reply verifiquem o mesmo caminho, em comparação com os serviços *TCP* e *UDP*. Contudo verificou-se tempo inferiores. Esta diferença nos resultados deve estar relacionada com uma eventual aumento da utilização do *CPU* no momento em que foram realizados os testes pois a variação dos valores quer na fase de comunicação quer na fase de decomposição é maior na experiência de activação de um serviço *ICMP*. O facto de não considerar a activação do serviço *ARP* também pode contribuir para a diferença verificada.

Relativamente ao aumento da topologia, verifica-se que os serviços apresentam um comportamento aproximadamente linear e proporcional com o aumento da topologia, verificando que com a duplicação da topologia o tempo de activação duplica aproximadamente.

5.4 Teste à Arquitectura Geral da Solução

Nesta secção iremos estudar o desempenho da arquitectura da solução geral de controlo da rede *Openflow* com a integração dos seus diversos módulos.

5.4.1 *Testbed*

A testebed utilizada para a seguinte experiência corresponderá à topologia de rede representada pela figura 4.1, constituída por 4 nós e cujas características de *hardware* e *software* se encontram resumidas na tabela 4.1.

5.4.2 Metodologia

A metodologia da experiência pode ser divida em duas grandes partes: a primeira em que se estudará o desempenho da interação entre o módulo *service manager* e o módulo *activator*, e a segunda que analisa o comportamento da arquitectura e problemas na rede.

A primeira parte da experiência consistirá na activação de um conjunto de pedidos de serviços de conectividade sobre a rede, alterando ao longo da experiência o número de fluxos nos quais cada pedido irá ser decomposto. A segunda parte da experiência, analisará o tempo de resposta da arquitectura a problemas na rede, mais concretamente, a perda de um *link* e detecção de um novo *link* no substrato, que desencadeará no módulo *service manager* uma reoptimização dos fluxos presentes na rede, como se pode observar na figura 5.28.

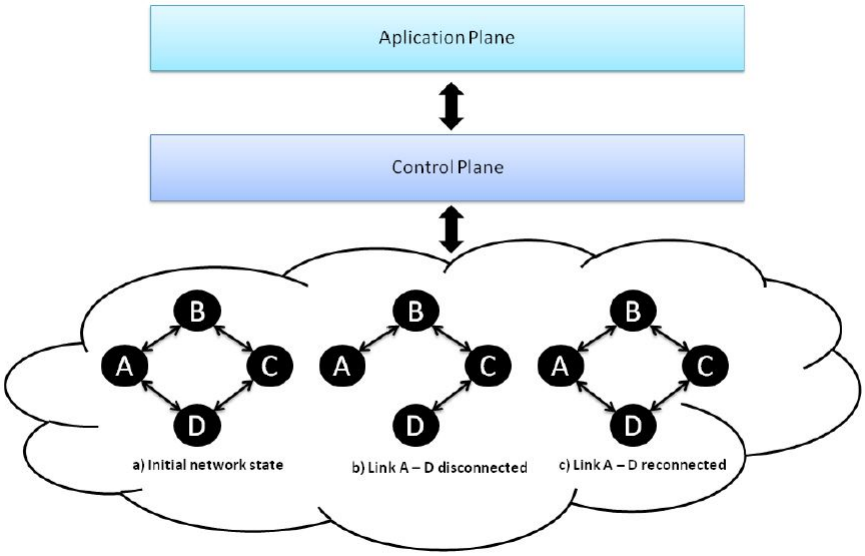

Figura 5.28: Metodologia utilizada para análise da arquitectura geral

Assim sendo estudar-se-ão duas fases diferentes nos testes propostos para a arquitectura geral. Na primeira fase, considera-se que para a activação dos fluxos, a fonte de enviou se encontra no nó A, a partir do qual serão enviados pedidos de activação de um serviço para o nó D, que será decomposto num conjunto de fluxos. Seguidamente efectua-se o corte de ligação entre o nó A e o nó D, activando assim um alarme no módulo *monitoring* que notificará o módulo *service manager* do problema presente na rede, provocando uma

reoptimização na rede dos fluxos afectados. Depois efectua-se a reposição da ligação entre o nó A e o nó D que desencadeará um conjunto de acções semelhantes àquelas provocadas na perda de um *link*.

Numa segunda fase, considera-se para a activação dos fluxos, que a fonte de envio se encontra no nó A, a partir do qual serão enviados pedidos de activação de um serviço para os nós B e D, que será decomposto num conjunto de fluxos. Seguidamente efectua-se o corte de ligação entre o nó A e o nó D, como se pode observar na figura 5.28, activando assim um alarme no módulo *monitoring* que notificará o módulo *service manager* do problema presente na rede, provocando uma reoptimização na rede dos fluxos afectados. Depois efectuar-se-á a reposição da ligação entre o nó A e o nó D, como se pode observar na figura 5.28, que desencadeará um conjunto de acções semelhantes às provocadas na perda de um *link*.

Nestes testes são analisados os tempos de activação do serviço na rede por parte do módulo *activator*, o tempo de detecção da perda do *link* por parte do módulo *monitoring* e o tempo de reoptimização por parte do módulo *service manager*.

5.4.3 Activação de serviço entre o nó A e o nó D

Esta experiência testa a activação de um serviço, com dois elementos, entre o nó A e o nó D, da figura 5.28, que se irá decompor na activação de um fluxo na rede. A experiência repete-se 10 vezes, obtendo-se os seguintes resultados com um intervalo de confiança de 95 %, que são apresentados na tabela 5.5.

Resultados Obtidos da Arquitectura da Solução			
Módulo	**Fase de Activação (s)**	**Fase de Reconfiguração por perda de *link* (s)**	**Fase de Reconfiguração por detecção de um novo *link* (s)**
Módulo *Monitoring*	——–	0.104 ± 0.010	3.10± 0.110
Módulo *Service Manager*	0.320 ±0.063	0.189± 0.043	0.174± 0.010
Módulo *Activator*	0.045 ±0.005	0.089 ± 0.005	0.085± 0.005
Total	0.365 ±0.063	0.382 ± 0.100	3.36 ± 0.010

Tabela 5.5: A activação de um serviço *UDP* que é decomposto em 1 fluxo

Por observação dos valores apresentados na tabela 5.5 pode-se concluir que o tempo total de activação dos serviços na rede, isto é, desde o momento em que é feito o pedido até a informação ficar disponível na rede, demora aproximadamente 0.365 segundos. O tempo total de reagir a uma alteração da rede pela detecção da perda de um *link* é de 0.382 segundos, dos quais 0.104 segundos corresponde ao tempo de detectar e notificar a perda de um *link* por parte do módulo de *monitoring* ao módulo *service manager*, e os restantes 0.278 segundos corresponde ao tempo de alterar o caminho seguido pelo fluxo na rede, por parte do módulo de *service manager* e o módulo *activator*, a fim de continuar a garantir o funcionamento do serviço na rede. Os resultados obtidos encontram-se dentro da gama de valores apresentados por cada um dos módulos da arquitectura, apresentando sempre um aumento no tempo que se prende com o tempo de estabelecimento da ligação para a comunicação entre os módulos.

Finalmente, o tempo de reacção por parte da arquitectura à detecção de um novo *link* na rede é de 3.36 segundos, dos quais 3.10 segundos correspondem ao tempo de detectar e notificar a presença de um novo *link* na rede por parte do módulo *monitoring* ao módulo *service manager*. Os restantes 0.226 segundos correspondem ao tempo de reoptimizar os fluxos presentes na rede. Verifica-se um aumento dos tempos conseguidos na detecção da perda de *link*, o qual se vê duplicado. Este facto deverá estar relacionado com o tempo de estabilização das ligações entre módulos para efectuar a sua comunicação. Relativamente à detecção de um

novo *link*, só há um ligeiro aumento dos valores que devem estar relacionados com o tempo de ligação para o estabelecimento da comunicação entre módulos. Outro factor que também pode contribuir para o aumento deste tempo é a utilização do *CPU*, no instante em que está a ser realizado o teste, no qual estão a correr as máquinas. Por isso pode-se considerar que o tempo de detecção de perda de um *link* e de detecção de um *link* é aceitável, de acordo aos tempos verificados nas secções 5.2.2.2 e 5.2.3.2.

5.4.4 Activação de serviço entre o nó A e os nós B e D

A activação de um serviço, com 5 elementos, entre o nó A e os nós B e D, da figura 5.28, será decomposta na activação de quatro fluxos na rede, dois entre A e B e outros dois entre A e D. Repetiu-se a experiência 10 vezes, obtendo-se os seguintes resultados com um intervalo de confiança de 95 % que são apresentados na tabela 5.6

Resultados Obtidos da Arquitectura da Solução			
Módulo	**Fase de Activação (s)**	**Fase de Reconfiguração por perda de *link* (s)**	**Fase de Reconfiguração por detecção de um novo *link* (s)**
Módulo *Monitoring*	——–	0.105 ± 0.010	3.13± 0.150
Módulo *Service Manager*	0.714 ±0.050	0.372± 0.053	0.402± 0.083
Módulo *Activator*	0.192 ±0.013	0.181 ± 0.009	0.180± 0.011
Total	0.906 ±0.050	0.658 ± 0.053	3.71 ± 0.083

Tabela 5.6: Activação de um serviço *UDP* que é decomposto em 2 fluxos

Por observação dos valores apresentados na tabela 5.6 pode-se concluir que o tempo total de activação dos serviços na rede demora aproximadamente 0.906 segundos. O tempo total de reagir a uma alteração da rede pela detecção da perda de um *link* é de 0.658 segundos, dos quais 0.105 segundos é o tempo de detectar e notificar a perda de um *link* por parte do módulo de *monitoring* ao módulo *service manager*, e 0.553 segundos é o tempo de alterar o caminho seguido pelo fluxos na rede, por parte do módulo de *service manager* e o módulo *activator*, a fim de garantir o funcionamento do serviço na rede.

Finalmente, o tempo para reagir à detecção de um novo *link* na rede é de 3.71 segundos, dos quais 3.13 segundos correspondem ao tempo de detectar e notificar a presença de um novo *link* na rede por parte do módulo *monitoring* ao módulo *service manager*. Os restantes 0.582 segundos correspondem ao tempo de reoptimizar os fluxos, por parte do módulo de *service manager* e o módulo *activator*, presentes na rede.

Como era expectável, com o aumento do número de fluxos a activar na rede, os tempos também aumentaram e na mesma proporção. Neste caso o serviço é decomposto em 4 fluxos que será decomposto na activação de 8 regras, quadriplicando aproximadamente assim os tempos de activação serviço no módulo *activator* em relação à experiência anterior. Por outro lado, no momento da reconfiguração será preciso alterar 2 fluxos, duplicando assim os tempos de reconfiguração dos módulos da arquitectura.

5.4.5 Activação de serviço entre o nó A e os nós B e D

A activação de um serviço, com 20 elementos, entre o nó A e os nós B e D, da figura 5.28, será decomposta na activação de 19 fluxos na rede, 3 no nó A, 4 entre o nó A e o nó B e 12 entre o nó entre A e o nó D. Repetiu-se a experiência 10 vezes, obtendo-se os seguintes resultados com um intervalo de confiança de 95 % que são apresentados na tabela 5.6

Resultados Obtidos da Arquitectura da Solução			
Módulo	**Fase de Activação (s)**	**Fase de Reconfiguração por perda de *link* (s)**	**Fase de Reconfiguração por detecção de um novo *link* (s)**
Módulo *Monitoring*	——–	0.119 ± 0.010	3.18± 0.116
Módulo *Service Manager*	2.94 ±0.092	2.39± 0.202	2.31± 0.061
Módulo *Activator*	1.14 ±0.012	1.36 ± 0.023	1.27± 0.030
Total	4.08 ±0.092	3.87 ± 0.202	6.76 ± 0.061

Tabela 5.7: Activação de um serviço *UDP* que é decomposto em 19 fluxos

Por observação dos valores apresentados na tabela 5.7 pode-se concluir que o tempo total de activação dos serviços na rede demora aproximadamente 4.08 segundos.

O tempo total de reagir a uma alteração da rede pela detecção da perda de um *link* é de 3.87 segundos, dos quais 0.119 segundos correspondem ao tempo de detectar e notificar a perda de um *link* por parte do módulo de *monitoring* ao módulo *service manager*, e 2.98 segundos é o tempo de alterar o caminho seguido pelos fluxos na rede, por parte do módulo de *service manager* e o módulo *activator*, a fim de continuar garantir o funcionamento do serviço na rede.

Finalmente o tempo de reagir por parte da arquitectura a detecção de um novo *link* na rede é de 6.76 segundos, dos quais 3.18 segundos correspondem ao tempo de detectar e notificar a presença de um novo *link* na rede por parte do módulo *monitoring* ao módulo *service manager*. Os restantes 3.58 segundos correspondem ao tempo de reoptimizar os fluxos presentes na rede por parte dos módulo de *service manager* e *activator*.

Com o aumento do número de fluxos a activar e a reoptimizar na rede, verificou-se um aumento nos tempos de activação e reconfiguração proporcional ao número de fluxos a activar e reconfigurar. Neste caso o serviço é decomposto em 19 fluxos ficando os tempos de activação serviço no módulo *activator* 5 vezes maiores, aproximadamente. Por outro lado, no momento da reconfiguração será preciso alterar 6 fluxos, ficando assim os tempos de reconfiguração 6 vezes maior. Se analisarmos os valores com a primeira experiência verifica-se também um aumento proporcional nos tempos de activação e reoptimização dos fluxos.

5.4.6 Resultados Sumários

Nesta secção apresentamos os resultados finais obtidos para todas as experiências realizadas para a arquitectura da solução proposta, como se pode observar na tabela 5.8.

Resultados Globais para os vários serviços a activar na rede			
Serviço	**Tempo de Activação (s)**	**Tempo da reconfiguração por perda de *link***	**Tempo da reconfiguração pela detecção de um novo *link***
Serviço de 1 fluxo	0.365 ±0.063	0.382± 0.043	3.36± 0.110
Serviço de 2 fluxos	0.906 ± 0.050	0.658± 0.053	3.71± 0.150
Serviço de 19 fluxos	4.08 ± 0.092	3.87 ± 0.202	6.76 ± 0.116

Tabela 5.8: Tabela dos resultados globais para os vários serviços a activar na rede

O tempo de activação do serviço aumenta proporcionalmente com o aumento do número de fluxos que o constituem, ou seja, um serviço com dois fluxos demorará aproximadamente o dobro do tempo que um serviço com um fluxo. Note-se que no caso do serviço que é decomposto em 19 fluxos a proporcionalidade directa não

se verifica, devido à existência de três fluxos desse serviço serem activos num único *switch*, apresentando por tal motivo um valor ligeramente inf ao esperado.

Outra conclusão que se pode observar é que o tempo de reconfiguração quer pela perda de *link* quer pela adição de um novo *link* apresenta um comportamento linear e que aumenta com o número de fluxos activos a reconfigurar na rede. Nota-se que o tempo de reconfiguração pela perda de *link* vê-se mais prejudicado, que o da detecção de um novo *link* com o aumento do número de fluxos presentes na rede, pois a reconfiguração implica que os fluxos passem por um número maior de *switches* que no caso da detecção de um novo *link*, onde haverá uma reoptimização de caminhos que se traduzirá em fluxos que passam por um menor número de *switches*.

Verificou-se também um aumento no tempo gasto pelo módulo *monitoring* para notificar, neste caso, ao *service manager* a perda de *link* relativamente aos valores apresentados para a perda de um *link* na secção 5.2.1. Este aumento prende-se com um aumento verificado no tempo de efectuar a comunicação *TCP* entre o módulo *monitoring* e o *Floodlight* e o módulo *monitoring* e o módulo *service manager*. Outra das razões encontradas que justificam a diferença de valores prende-se com a utilização do *CPU* no momento das experiências que puderam também ter afectado os valores da comunicação. Outro factor relevante que podemos concluir da experiência é que com o aumento de número de fluxos presentes nos *switches* o tempo de detecção da perda ou adição de um novo *link* aumenta, pelas razões já explicadas. Por outro lado, a diferença que se encontra entre o tempo de detectar a perda de um *link* com o tempo de detectar um novo *link* está relacionado com a reconfiguração interna da interface, necessária por parte da máquina que emula o *OpenVswitch*, como foi explicado na secção 5.2.2.2.

5.5 Conclusão

Neste capítulo avaliámos o desempenho da solução proposta, submetendo a um conjunto de testes e experiências cada um dos módulos desenvolvidos no âmbito desta Dissertação e que constituem o protótipo da solução proposta.

Relativamente às experiências realizadas para medir o desempenho do módulo *monitoring*, isto é, a medição do tempo de resposta por parte do módulo para detectar a perda de um *link*, obtiveram-se bons resultados que indicam que o módulo demora, aproximadamente, 19 ms a detectar a perda de *link*, 26 ms a enviar a notificação de alteração na topologia aos módulos previamente susbscritos, num total de 41 milissegundos. Para a outra experiência proposta para medir o desempenho do módulo, isto é, a detecção de um *link* na rede, obteve-se que o módulo demora 2.78 segundos a detectar a presença de um novo *link* na rede, dispendendo 20 ms a enviar aos módulos previamente subscritos, totalizando assim 2.81 segundos no processo de detectar a presença de um novo *link* na rede. As diferenças verificadas no tempo de resposta a notificar a presença de um novo *link* e a perda de um *link* prende-se com a metodologia utilizada no teste, visto que, sempre que a interface é activada, existe um processo de reconfiguração interna por parte do computador que emula o *OpenVswitch*, que consequentemente notifica o *OpenVswitch*, o qual finalmente irá notificar *Floodlight* a presença de um novo *link*. Outra das razões que pode contribuir para o aumento dos tempos é o tempo de estabelecimento da ligação *TCP* para realizar a comunicação entre os módulos.

Relativamente aos testes realizados para o módulo *activator*, verificou-se que o processo de activação de um fluxo na rede depende de duas componentes que são: a fase de decomposição, responsável por fazer a tradução de fluxos para regras, e a fase de comunicação, que é a encarregue de efectuar a comunicação com o controlador e efectuar a correspondente activação na rede. Também se verificou que este processo apresenta

um comportamento aproximadamente linear com o aumento número de fluxos presentes na rede, e que é proporcional ao número de dispositivos presentes na rede. Outra das ilações que estes testes permitiram retirar foi que quer a fase de decomposição como a fase de comunicação apresentam um comportamento linear e que aumenta com o aumento de número de fluxos presentes na rede, sendo a fase de decomposição a que maior incidência apresenta nos valores finais do tempo de activação.

Por outro lado verificou-se que os serviços de comunicação *TCP e UDP* apresentam um comportamento semelhante em relação ao tempo dispendido para activação do mesmo, enquanto que o serviço *ICMP* apresenta um comportamento proporcional aos verificados ao serviço *UDP e TCP*, pois de acordo com a especificação *Openflow* e a implementação efectuada, a activação deste serviço implica a activação do dobro do número de regras de um serviço *TCP* e *UDP*, já que implica a activação do serviço *ARP*.

Finalmente, sobre o teste da arquitectura em geral verificou-se que o tempo de activação do serviço aumenta proporcionalmente com o aumento do número de fluxos que o constituem. O tempo de reconfiguração, quer pela perda de *link* quer pela adição de um novo *link* apresenta um comportamento aproximadamente linear e que aumenta com o número de fluxos activos na rede. O tempo de reconfiguração pela perda de *link* vê-se mais prejudicado que o a detecção de um novo *link*, com o aumento do número de fluxos presentes na rede, pois a reconfiguração implica que os fluxos passem por um número maior de *switches*, que no caso da detecção de um novo *link*, haverá uma reoptimização de caminhos que se traduzirá em fluxos que passam por um menor número de *switches*.

Capítulo 6

Conclusões

As *SDNs* e o protocolo *Openflow* terão forte preponderância na Internet de futuro, em especial na sua arquitectura e modo de funcionamento. O número de dispositivos que podem ligar-se à Internet tem aumentado consideravelmente e prevê-se que este número continue a aumentar. No passado, os dispositivos com ligação à Internet eram principalmente *PCs*; hoje em dia telemóveis, televisões, impressoras, carros, também fazem parte desta rede global. Actualmente a Internet já não é utilizada apenas por seres humanos para comunicar, mas também para que as máquinas comuniquem entre si, provocando uma alteração nos padrões de tráfego da rede. Esta situação impulsionou o desenvolvimento de soluções para darem resposta a este novo modo de comunicação, que no início conseguia responder aos requisitos previstos, mas que com o aumento do número dispositivos da rede e dos seus requisitos, tem dificultado o suporte dos mesmos. Surge assim o conceito das *SDNs* e o protocolo *Openflow* como solução para a ossificação da rede, prevendo uma alteração da arquitectura da rede mediante a separação do plano de dados e o plano de controlo.

Algumas funcionalidades das *SDNs* e do protocolo *Openflow* já foram testadas com bons resultados, como por exemplo, os *datacenters* da Google e o desenvolvimento de *hardware switches* por parte da HP, entre outros. Contudo, a evolução das *SDNs* não tem seguido o ritmo que os gestores de redes esperavam que tivessem, pois factores como a unificação de uma especificação e de segurança tem travado o seu crescimento e aceitação por parte dos fornecedores de serviço, para uma inversão a longo prazo das mesmas como novo paradigma da arquitectura da Internet.

Alguns esforços têm sido feitos, nomeadamente a especificação do protocolo *Openflow* tem vindo a sofrer alterações contínuas com o objectivo de responder aos requisitos dos clientes, e o número de empresas que se juntam ao projecto tem vindo a aumentar. No entanto, o desenvolvimento das ferramentas para aplicação e testes não tem seguido a mesma evolução, como por exemplo, controladores e *hardware* que suportem algumas das últimas especificações do protocolo.

As SDNs apresentam-se como uma tecnologia emergente, nas quais é preciso continuar a trabalhar e realizar experiências, com o objectivo de perceber as suas limitações e o seu desempenho, de modo a torná-las mais seguras e robustas para conseguir dar resposta aos requisitos actuais da Internet.

O trabalho desenvolvido ao longo desta Dissertação centrou-se na construção de um demonstrador de rede *Openflow* capaz de testar as funcionalidades que as *SDNs* apresentam. Por tal motivo, foi necessário implementar uma *testbed* que permitisse a construção de uma rede baseada nos princípios das redes definidas por *software* e que utiliza o protocolo normalizado *Openflow*. O demonstrador apresenta um papel preponderante na camada de controlo, tendo sido necessário o desenvolvimento de um gestor de rede responsável pelo controlo da rede *Openflow*. Este gestor é constituído por 4 módulos, dos quais dois foram alvo de estudo e desenvolvimento

nesta Dissertação. O primeiro destes dois módulos é o módulo *monitoring* responsável pela monitorização da rede, isto é, verificar o seu estado e notificar possíveis problemas da mesma. O segundo módulo é o módulo *activator* responsável pela activação dos fluxos correspondentes aos serviços que se pretendem activar na rede. Finalmente foram desenvolvidos mecanismos que garantem a comunicação entre os módulos.

Este demonstrador foi testado em vários cenários para verificar os princípios das SDNs, a separação do plano de dados e de controlo, centralizando a inteligência da rede na camada de controlo. Também se conseguiu verificar a conectividade entre os dispositivos da rede, assim como efectuar alterações nos padrões de tráfego mediante, e unicamente, programação do plano de controlo.

Por outro lado, sobre os testes de desempenho podemos concluir que os resultados obtidos para o módulo de *monitoring*, mais concretamente a detecção e aviso da perda de *link* da rede se encontram no limiar do aceitável das redes de hoje. Relativamente ao módulo *activator*, verificou-se que o tempo de activação dos fluxos apresenta um comportamento aproximadamente linear com o aumento do número de fluxos presente na rede e ao aumento da topologia. Observou-se ainda que este tempo apresenta a contribuição de duas componentes, que são a componente de decomposição de fluxos em regras e a componente relativa ao tempo de comunicação para activação da regra na rede, sendo a primeira a que apresenta uma maior contribuição no tempo final de activação do fluxo na rede.

Por isso podemos dizer que atingimos os objectivos propostos no inicio deste trabalho, isto é, conseguiu-se a construção de uma *testbed* baseada na arquitectura das *SDNs*, e que implementa o protocolo *Openflow*, e de um gestor de rede constituído por 4 sub-módulos, apresentando bons resultados e que serviram de motivação para continuar a apostar nesta área inovadora.

Relativamente ao trabalho futuro, este deve numa primeira fase centrar-se no desenvolvimento de uma base de dados única para o armazenamento de todas as informações necessárias, por parte dos diversos módulos do gestor de rede. Seguidamente, deve efectuar-se o desenvolvimento de programas de teste que permitam verificar os alarmes que foram desenvolvidos, assim como desenvolver outro tipo de alarmes que permitam ter o maior número de informações da rede, a partir dos quais seja possível detectar os problemas que as redes de hoje apresentam. Depois de incluídas estas melhorias, deve prosseguir-se para a integração e implementação de uma das últimas especificações *Openflow* na plataforma, sempre e quando a evolução das ferramentas de aplicação, isto é, controladores e *hardware* assim o permitam. Seguidamente, é necessário efectuar o estudo e análise das novas funcionalidades que esta plataforma apresenta.

Num outro âmbito, é importante abordar a utilização das *SDN* para a aceder a serviços *cloud*, como por exemplo o serviço *Meo*, e realizá-lo de uma forma integrada. Finalmente, a plataforma deve ser dotada de parâmetros e mecanismos de QoS.

Bibliografia

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] Dan Talayco David Erickson Ben Pfaff, Brandon Heller. Openflow switch specification, version 1.0.0. `http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf`, 2009.

[3] Dan Talayco David Erickson Ben Pfaff, Brandon Heller. Openflow switch specification. `https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf`, 2012.

[4] Bugenhagen M. Khan W. Chiosi M. Clarke D. Cui C. Damker H. Delisle D. Demaria E. Deng H. Fargano M. Feger J. Fukui M. Ivano G. Kolias C. Lopez D. Loudier Q. Manzalini A. Matsuzaki T. Michel U. Minerva R. Ogaki K. Reid A. Ruhl F. Salguero F. J. R. Sen P. Shimano K. Benitez, J. and Willis. Network functions virtualisation an introduction, benefits, enablers, challenges call for action. 2012. SDN and OpenFlow World Congress, Darmstadt-Germany.

[5] Chad Berndtson. Hp networking: Openflow support across the board in 2013. `http://www.crn.com/news/networking/240008259/hp-networking-openflow-support-across-the-board-in-2013.htm`, 2013.

[6] Andrea Bianco, Robert Birke, Luca Giraudo, and Manuel Palacin. Openflow switching: Data plane performance. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[7] JM BONIN and A NOGUEIRA. Aplicabilidade de cloud computing computaÃ§Ã£o em nuvem como fator de reduÃ§Ã£o de custos nas diversas areas de negocios. *Programa de Pos-GraduaÃ§Ã£o em InformÃ¡tica–Universidade Federal de Santa Maria (UFSM), ECAUSP. Santa Maria–RS-Brasil*, 2010.

[8] Andrew T Campbell, Irene Katzela, Kazuho Miki, and John Vicente. Open signaling for atm, internet and mobile networks (opensig'98). *ACM SIGCOMM Computer Communication Review*, 29(1):97–108, 1999.

[9] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[10] Gaetano Catalli. Open vswitch: performance improvement and porting to freebsd. `http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/poster_001.pdf`, 2011.

[11] NM Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[12] Saurav Das, Guru Parulkar, Nick McKeown, Preeti Singh, Daniel Getachew, and Lyndon Ong. Packet and circuit network convergence with openflow. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pages 1–3. IEEE, 2010.

[13] David Davis. Introduction to software-defined networking (sdn). http://www.windowsnetworking.com/articles-tutorials/cloud-computing/introduction-software-defined-networking-sdn.html, 2013.

[14] Comer Douglas E. *Computer Networks and Internets*. Prentice-Hall, 2001.

[15] Jim Duffy. Cisco ends the sdn suspense with cisco one. http://news.techworld.com/networking/3364097/cisco-ends-sdn-suspense-with-cisco-one/, 2012.

[16] Diego Henrique Duque. Redes definidas por software. http://br.monografias.com/trabalhos3/redes-definidas-software/redes-definidas-software.shtml, 2012.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), 1999. Updated by RFCs 2817, 5785, 6266.

[18] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.

[19] Open Networking Foundation. Whitepaper software-defined networking the new norm for networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012.

[20] Norberto Gallego. Cisco ends the sdn suspense with cisco one. http://www.norbertogallego.com/cisco-protege-su-tesoro/2012/07/05/, 2012.

[21] Georg Gruetter. The internet of things for the rest of us. http://blog.bosch-si.com/the-internet-of-things-for-the-rest-of-us/, 2012. Bosch Software Innovations, Blogging Internet of Things - Technology inspiring a connected life.

[22] Dorgival Guedes, L Vieira, M Vieira, Henrique Rodrigues, and R Nunes. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, pages 160–210, 2012.

[23] Shimonishi Hideyuki and Shuji Ishii. Virtualized network infrastructure using openflow. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pages 74–79. IEEE, 2010.

[24] Fred Hsu, M Salman Malik, and Soudeh Ghorbani. Openflow as a service. https://wiki.engr.illinois.edu/download/attachments/197298167/CloudFlow-3rd.pdf?version=6&modificationDate=1336587605000, 2012.

[25] Carapinha Jorge and Jiménez Javier. Network virtualization: a view from the bottom. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 73–80. ACM, 2009.

[26] Organization JSON. Introducing json. http://www.json.org/, 2010.

[27] Allen Kevin. Pertino project. http://pertino.com/utm_source=pressrelease&utm_campaign=launchrelease, 2013.

[28] Georgia Kontesidou and Kyriakos Zarifis. *Openflow Virtual Networking: A Flow-Based Network VirtualizationArchitecture*. PhD thesis, KTH, 2009.

[29] Joseph Kovar. Pertino introduces cloud-based sdn service. http://www.crn.com/news/networking/240148381/pertino-introduces-cloud-based-sdn-service.htmpgno=1, 2013.

[30] Whitepaper Linux Foundation. The project. http://www.opendaylight.org/publications/opendaylight-open-source-community-and-meritocracy-software-defined-networking, 2011. Whitepaper.

[31] Carlos Alberto Braz Macapuna. Openflow e nox: Propostas para experimentaçao de novas tecnologias de rede. http://www.macapuna.com.br/index/phocadownload/userupload/trabalhos-academicos/Artigos-disciplina/main-ofnox.pdf, 2009.

[32] Andre Fernando Uebe Mansur, Samantha Silva Gomes, Arilise Moraes de Almeida Lopes, and MCB Biazus. Novos rumos para a informÃ¡tica na educaÃ§Ã£o pelo uso da computaÃ§Ã£o em nuvem (cloud education): Um estudo de caso do google apps. In *Foz do Iguacu: Anais do XVI Congresso Internacional ABED de EducaÃ§Ã£o a Distancia*, 2010.

[33] GutiÃ©rrez Manuel. Historia da internet. http://www.tufuncion.com/historia_internet, 2006.

[34] Filippetti Marco AurÃ©lio. *Guia Completo de Estudo CCNA*. LACALLE, 2010.

[35] Aguirre Higuita Maria Camila. Internet y red. http://www.monografias.com/trabajos14/internetyred/internetyred.shtml, 2003.

[36] Murphy McCauley. Nox project. http://www.noxrepo.org/, 2013.

[37] Shamus McGillicuddy. Respuesta de cisco sdn: Redes programables, no openflow. http://searchdatacenter.techtarget.com/es/noticias/2240173303/Respuesta-de-Cisco-SDN-Redes-programables-no-OpenFlow, 2012.

[38] Nick McKeown. How sdn will shape networking. http://mvdirona.com/jrh/talksandpapers/nickmckeown_on%20summit%20nickm%2010%202011.pdf, 2010.

[39] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[40] Ricardo André Martins Mendes. Suporte de gestão de plataforma ims baseado em netconf. Master's thesis, Universidade de Aveiro, 2010.

[41] Marc Mendonça, Bruno Nunes Astuto, Xuan Nam Nguyen, Katia Obraczka, Thierry Turletti, et al. A survey of software-defined networking: Past, present, and future of programmable networks. http://hal.archives-ouvertes.fr/docs/00/82/50/87/PDF/SDN_survey.pdf, 2013.

[42] Marcelo DD Moreira, Natalia C Fernandes, LHMK Costa, and OCMB Duarte. Internet do futuro: Um novo horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, 2009:1–59, 2009.

[43] Timothy Prickett Morgan. cisco onesdn openflow openstack. http://www.theregister.co.uk/2012/06/14/cisco_one_sdn_openflow_openstack/, 2012.

[44] Rackspace Nasa. Quantum. https://wiki.openstack.org/wiki/Quantum, 2013.

[45] Nogueira, João Pedro Brites Ferreira. Demonstração de criação de redes virtuais no âmbito do operador. Master's thesis, Universidade de Aveiro, 2010.

[46] Whitepaper Intel Organization. Open, simplified networking based on sdn and network functions virtualization. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/sdn-part-1-secured.pdf, 2013.

[47] Bruno Miguel Sendas Parreira. Integração da cloud com rede na perspectiva de operador. Master's thesis, Universidade de Aveiro, 2012.

[48] Ivan Pepelnjak. Nicira, bigswitch, nec, openflow and sdn. http://blog.ioshints.info/2012/02/nicira-bigswitch-nec-openflow-and-sdn.html, 2013.

[49] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtual switching in an era of advanced edges. In *2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES), ITC*, volume 22, 2010.

[50] Ben Pfaff. Why open vswitch? `http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=WHY-OVS;hb=HEAD`, 2012.

[51] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[52] Riccardo, Zoran Despotovic, Guerzoni, Riccardo Trivisonno, Ishan Vaishnavi, Artur Hecker, and Sergio Beker. Nfv and sdn in future carrier networks. ETSI workshop on Future Networks, 2013.

[53] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O Reilly, 2007.

[54] Alex Rodriguez. Restful web services: The basics. IBM Corporation, 2008.

[55] Sergio Rodríguez Santamaría. Mecanismos de control de las comunicaciones en la internet del futuro a través de openflow. `http://repositorio.unican.es/xmlui/bitstream/handle/10902/1165/Sergio%20Rodriguez%20Santamaria.pdf?sequence=1`, 2012.

[56] Guillermo Romero de Tejada Muntaner. *Evaluation of OpenFlow Controllers*. PhD thesis, KTH, 2012.

[57] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. `http://www.cpqd.com.br/cadernosdetecnologia/Vol7_N1_jul2010_jun2011/pdf/artigo6.pdf`, 2010.

[58] Fidel Salgueiro. Sdn (software defined networking): Las redes definidas por software fidel salgueiro. `http://fidelsalgueiro.blogspot.pt/`, 2012.

[59] Van Dijk Sandra. Virtualizacion que es. `http://cioperu.pe/articulo/8177/virtualizacion101-que-es-la-virtualizacion/`, 2011.

[60] Sharon, Dino Randy Katz, Barkai, Farinacci, and David Meyer. Software defined flow-mapping for scaling virtualized network functions. *OpenFlow World Congress*, 2012.

[61] Rob Sherwood, Glen Gibb, K-K Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[62] William Stallings. The internet protocol journal software-defined networks and openflow. `http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_16-1/161_sdn.html`, 2013.

[63] Sunay Tripathi. Of controllers and why nicira had to do a deal (part iii: Sdn and openflow enabling network virtualization in the cloud). `http://pluribusnetworks.com/blog/of-controllers-and-why-nicira-had-to-do-a-deal-part-3-sdn-openflow/`, 2012.

[64] Parraga Jason Wang, Kuang-Ching. Project floodlight. `http://www.projectfloodlight.org/floodlight/`, 2013.