Next Generation Networks - The Technologies and Enablers

Author: Anna Sofía Bidot García
Supervisor: Manzoor A. Khan

A thesis submitted in fulfilment of the requirements for the degree of Master of Telecommunications Engineering

IV Fakultät Elektrotechnik und Informatik
DAI-Labor

May 2015
Declaration of Authorship

I, Anna Sofía Bidot García, declare that this thesis titled, 'Next Generation Networks -The Technologies and Enablers ' and the work presented in it are my own. I confirm that:

■ This work was done wholly or mainly while in candidature for a research degree at this University.

■ Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

■ Where I have consulted the published work of others, this is always clearly attributed.

■ Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

■ I have acknowledged all main sources of help.

■ Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:
Abstract

While consumers are currently yearning for 4G smartphones, laboratories and some of the biggest companies are already working to develop its successor: the 5th generation networks. 5G will alter current technologies as we know them today. Software Defined Networking (SDN) will be the architecture used to deploy 5G. This thesis introduces the reader to the world of SDN. SDN breaks away from the conventional model of network, decoupling the network infrastructure from the applications and services it provides. The global network control in SDN is achieved by centralizing the logic function control plane. An element called 'Controller' implements the behavior rules of the switching elements.

Essentially, the objective of this paper is to present an overview of both the challenges and the vision of future networks, along with the analytical explanations and the precise definitions of cloud computing, SDN and Network Function Virtualization (VTN). OpenStack, as a cloud-computing platform is also presented, as well as the OpenFlow Protocol and Mininet, a network emulator tool. Additionally, the controller OpenDaylight is utilized to exhibit the extensive capabilities and immense potential of SDN network.

In this thesis, the OpenDaylight project is illustrated, implying its current projects and its technical aspects. This is in conjunction with an in-depth view of the OpenDaylight Controller, where the Service Abstraction Layer (SAL) resides. As a result of the application oriented extensible north-bound architecture, the two Northbound APIs (OSGi services and RESTful web services) are accessible. Finally, a progressive system is delineated, which demonstrates the technique of writing an OpenDaylight application using all three different methods - either with API Driven SAL, Model Driven SAL or with RESTful web services-. Supplementary to this is a test-case called "Solving a DoS attack", in which the AD-SAL and REST API methods are utilized as solutions to the attack. This illustrates how SDN architecture, through the OpenDaylight Controller, is able to modify a network.
Acknowledgements

First of all, I would like to thank my family, my parents Heiner Bidot and Francisca García, my sister Emily Bidot and my aunts Heidi Bidot and Gundi Bidot. Without them, I wouldn’t be who I am today and I would never have gotten this far. Thank you so much for your constant support, and for always believing in me. I really appreciate your help throughout my life as a student, firstly in Mallorca, then in Barcelona and finally in Berlin.

My sincere thanks to my supervisor Dr.-Eng. Manzoor Ahmed Khan for his guidance and for introducing me to the fascinating Software Defined Networking world. I would like to thank him also for giving me the opportunity of working with him in the “Tel Gebäude”. Not only was it a great experience but also a pleasure to have one of the best views of Berlin.

Thanks also to professor Dr.-Eng. Juan Luis Gorricho, who agreed to be my supervisor in Barcelona. I really appreciate his help and his kind attention throughout my thesis.

A special thanks to Gary Beecher, who has been there for me, through ups and downs. He, despite not belonging to the science world, always helped and listened to me, giving me his best advice and motivation.

And finally, thank you to all of my friends. Those I met here in Berlin, with whom I enjoyed this amazing city. Thank you to my university colleagues in Barcelona, with whom I shared lots of laughs and fun as well as many hours of study. Also to my friends outside of university (Bernat, Cristina J., Cristina M., Laura, Loli, María, Miriam and Sandra) who, despite the time and long distance, have been always there for me.
Agradecimientos

Darle las gracias ante todo a mi familia, mis padres Heiner Bidot y Francisca García, a mi hermana Emily Bidot y a mis tíos Heidi Bidot y Gundi Bidot. Sin ellos no sería quien soy hoy ni habría llegado hasta donde he llegado. Mil gracias por el apoyo constante, por creer siempre en mí. Realmente aprecio la ayuda incondicional a lo largo de mi vida como estudiante, primero en Mallorca, luego en Barcelona y finalmente en Berlín.

Mis más sincero agradecimiento a mi supervisor Dr.-Ing. Manzoor Ahmed Khan, quien me ha guiado e introducido al fascinante mundo de las redes definidas por software. Me gustaría agradecerle también por la oportunidad que se me ha brindado de trabajar con él en "Tel Gebäude". No sólo ha sido una muy buena experiencia, sino también ha sido un placer poder disfrutar día a día de una de las mejores vistas de Berlín.

Gracias también al profesor Dr.-Ing. Juan Luis Gorricho, quien aceptó ser mi supervisor en Barcelona. Agradezco realmente su ayuda y su siempre amable atención durante el proyecto.

Agradecer especialmente a Gary Beecher, quien ha estado a mi lado tanto para los buenos momentos como para los no tan buenos. Él, aun no perteneciendo al mundo científico, ha sabido ayudarme y escucharme, motivándome y aconsejándome en todo momento.

Y por último, pero no menos importante, a todas mis amistades. Aquellas que he hecho aquí en Berlín y con las que he disfrutado de esta magnífica ciudad. A mis compañeros de carrera en Barcelona, con los que he compartido millones de risas e infinitas horas de estudio. Y agradecer también a mis amigos fuera de ella (Bernat, Cristina J., Cristina M., Laura, Loli, María, Miriam y Sandra) quienes a pesar del tiempo y la distancia han estado siempre a mi lado.
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Cloud Services</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>SDN basic Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>NFV vision Source: <a href="https://network-functions-virtualization.com">https://network-functions-virtualization.com</a></td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Horizon Dashboard</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Traditional architecture vs. SDN</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>SDN Layers</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>Switch Components</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Packet processing</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>From traditional networks to SDN</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>Mininet Source: <a href="http://mininet.org">http://mininet.org</a></td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Scenario 1 switch and 2 hosts</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>sudo mn command</td>
<td>40</td>
</tr>
<tr>
<td>4.9</td>
<td>nodes, dump and net commands</td>
<td>41</td>
</tr>
<tr>
<td>4.10</td>
<td>h1 ifconfig command</td>
<td>41</td>
</tr>
<tr>
<td>4.11</td>
<td>s1 ifconfig command</td>
<td>42</td>
</tr>
<tr>
<td>4.12</td>
<td>Network topology with 2 switches and 6 hosts</td>
<td>43</td>
</tr>
<tr>
<td>4.13</td>
<td>Virtual network topology example</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Technical Overview Source: <a href="http://www.opendaylight.org">http://www.opendaylight.org</a></td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Controller architecture</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Service Abstraction Layer Source: <a href="http://www.opendaylight.org">http://www.opendaylight.org</a></td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>From AD-SAL to MD-SAL Source: <a href="https://wiki.opendaylight.org">https://wiki.opendaylight.org</a> MD-SAL App Tutorial</td>
<td>52</td>
</tr>
<tr>
<td>6.1</td>
<td>Two ways of visualizing an application in ODL platform</td>
<td>55</td>
</tr>
<tr>
<td>6.2</td>
<td>Scenario used for AD-SAL application</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>pingall unreachable</td>
<td>62</td>
</tr>
<tr>
<td>6.4</td>
<td>Results</td>
<td>63</td>
</tr>
<tr>
<td>6.5</td>
<td>MD-SAL Software Architecture View Source: <a href="https://wiki.opendaylight.org">https://wiki.opendaylight.org</a> - MD-SAL App Tutorial</td>
<td>63</td>
</tr>
<tr>
<td>6.6</td>
<td>Building a Plug-in/Application Source Os2014-md-sal-tutorial.pdf</td>
<td>65</td>
</tr>
<tr>
<td>6.7</td>
<td>GET Network Topology AD-SAL</td>
<td>76</td>
</tr>
<tr>
<td>6.8</td>
<td>GET Flow Statistics AD-SAL</td>
<td>77</td>
</tr>
<tr>
<td>6.9</td>
<td>GET Switches AD-SAL</td>
<td>77</td>
</tr>
<tr>
<td>6.10</td>
<td>RESTCONF apidoc</td>
<td>78</td>
</tr>
<tr>
<td>6.11</td>
<td>Get Data RESTCONF</td>
<td>78</td>
</tr>
<tr>
<td>6.12</td>
<td>Flow tables before adding flow</td>
<td>80</td>
</tr>
</tbody>
</table>
6.13 PUT Flow ......................................................... 80
6.14 Results ......................................................... 80
List of Tables

2.1 Published specifications on NFV .................................................. 15
3.1 OpenStack Components ............................................................. 22
4.1 Controllers comparison .............................................................. 31
5.1 Important bundles OpenDaylight .................................................. 51
5.2 Differences between AD-SAL and MD-SAL Source: http://sdntutorials.com/difference-between-ad-sal-and-md-sal .................................................. 53
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration of Authorship</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Agradecimientos</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>Contents</td>
<td>viii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Scope</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Outline</td>
<td>2</td>
</tr>
<tr>
<td><strong>2 State of the Art Technologies</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Cloud Computing</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1 Cloud services</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1.1 Examples</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2 Deployment Models</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Software Defined Networking</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Network Functions Virtualization</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 NFV technology</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2 NFV definition</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2.1 European Telecommunications Standards Institute</td>
<td>13</td>
</tr>
<tr>
<td>2.3.2.2 Benefits of NFV</td>
<td>16</td>
</tr>
<tr>
<td><strong>3 OpenStack</strong></td>
<td>18</td>
</tr>
<tr>
<td>3.1 OpenStack Objective</td>
<td>18</td>
</tr>
<tr>
<td>3.2 What is OpenStack?</td>
<td>19</td>
</tr>
<tr>
<td>3.3 How to install multi nodes in OpenStack</td>
<td>23</td>
</tr>
</tbody>
</table>
# List of Tables

## 4 Software Defined Networking

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 SDN, an Introduction</td>
<td>27</td>
</tr>
<tr>
<td>4.1.1 Motivation to use SDN</td>
<td>28</td>
</tr>
<tr>
<td>4.1.2 Architecture</td>
<td>29</td>
</tr>
<tr>
<td>4.2 Controllers</td>
<td>30</td>
</tr>
<tr>
<td>4.2.1 NOX</td>
<td>32</td>
</tr>
<tr>
<td>4.2.2 POX</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3 Beacon</td>
<td>32</td>
</tr>
<tr>
<td>4.2.4 Floodlight</td>
<td>32</td>
</tr>
<tr>
<td>4.2.5 OpenDaylight</td>
<td>33</td>
</tr>
<tr>
<td>4.3 OpenFlow</td>
<td>33</td>
</tr>
<tr>
<td>4.3.1 OpenFlow Switch</td>
<td>34</td>
</tr>
<tr>
<td>4.3.2 SDN and OpenFlow</td>
<td>36</td>
</tr>
<tr>
<td>4.4 Mininet</td>
<td>37</td>
</tr>
<tr>
<td>4.4.1 Procedure</td>
<td>38</td>
</tr>
<tr>
<td>4.4.2 Study of a network topology</td>
<td>39</td>
</tr>
<tr>
<td>4.4.3 Python API Examples</td>
<td>42</td>
</tr>
<tr>
<td>4.4.4 Creating a virtual network topology</td>
<td>43</td>
</tr>
</tbody>
</table>

## 5 OpenDaylight

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 OpenDaylight, a first view</td>
<td>45</td>
</tr>
<tr>
<td>5.1.1 Current Projects, an overview</td>
<td>46</td>
</tr>
<tr>
<td>5.2 Technical aspects</td>
<td>47</td>
</tr>
<tr>
<td>5.3 Controller Project[21]</td>
<td>49</td>
</tr>
<tr>
<td>5.3.1 Service Abstraction Layer</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1.1 From AD-SAL to MD-SAL</td>
<td>51</td>
</tr>
</tbody>
</table>

## 6 OpenDaylight Application Development

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Background</td>
<td>54</td>
</tr>
<tr>
<td>6.2 Writing an Application using OSGi interfaces</td>
<td>56</td>
</tr>
<tr>
<td>6.2.1 AD-SAL Application Development</td>
<td>56</td>
</tr>
<tr>
<td>6.2.1.1 Scenario</td>
<td>56</td>
</tr>
<tr>
<td>6.2.1.2 IDE Setup</td>
<td>57</td>
</tr>
<tr>
<td>6.2.1.3 POM XML File</td>
<td>57</td>
</tr>
<tr>
<td>6.2.1.4 Activator Java</td>
<td>59</td>
</tr>
<tr>
<td>6.2.1.5 Plugin Code</td>
<td>59</td>
</tr>
<tr>
<td>6.2.1.6 Evaluation test</td>
<td>62</td>
</tr>
<tr>
<td>6.2.2 MD-SAL Application Development</td>
<td>63</td>
</tr>
<tr>
<td>6.2.2.1 POM XML file</td>
<td>64</td>
</tr>
<tr>
<td>6.2.2.2 Activator Java</td>
<td>64</td>
</tr>
<tr>
<td>6.2.2.3 Plugin Code Overview</td>
<td>65</td>
</tr>
<tr>
<td>6.2.2.4 Prepare the project</td>
<td>65</td>
</tr>
<tr>
<td>6.2.2.5 POM XML File</td>
<td>67</td>
</tr>
<tr>
<td>6.2.2.6 Karaf Feature</td>
<td>67</td>
</tr>
<tr>
<td>6.2.2.7 MD-SAL Model</td>
<td>69</td>
</tr>
<tr>
<td>6.2.2.8 Implementing the Provider Plugin</td>
<td>71</td>
</tr>
<tr>
<td>6.2.2.9 How Karaf knows about the Config Subsystem Dependency</td>
<td>73</td>
</tr>
</tbody>
</table>
6.2.2.10 Implementing Consumer Plugin ........................................ 74
6.3 Writing an Application using RESTful APIs .......................... 75
  6.3.1 AD-SAL REST API ................................................. 76
  6.3.2 MD-SAL RESTCONF API .......................................... 77
  6.3.3 How to add a flow .................................................. 78
    6.3.3.1 Plugin Code ................................................ 78
  6.3.4 Test the Code ..................................................... 79

7 Conclusions and Future Work ........................................... 82
Chapter 1

Introduction

1.1 Motivation

Although the deployment of 4th generation mobile networks has not yet been completed, a series of R & D initiatives have been launched to develop the technology Mobile 5th generation (5G), with the idea of commercializing this technology by 2020.

5G represents a paradigm shift in the mobile network design. It completely changes the current technology in order to support the flow requirements, latency and scalability necessary to meet extreme use cases such as augmented reality or the connection of billion devices.

The new generation of networks will transform how we live, and how we perceive the world today. It could be said that with 5G networks, we will live in a reality in which everybody and everything will be connected; not only to each other but also to every single device capable of gaining access to the internet (internet of things).

Companies such as Ericsson have already proved that the speed of a 5G network prototype has a 5Gbps data transfer rate, 50 times faster than the 100 Mbps fiber optic connection offered by operators. Here is where concepts such as Software Defined Networking (SDN) and Network Function Virtualization come in to play. With SDN architecture and NFV technology, it will be essential to enable telecommunications providers to create agile, scalable and efficient enough platforms. These platforms will be able to meet the requirements and demands of a hyper-fast network. Consequently, a new open network era will form.
In addition to these concepts, the OpenDaylight Open Source Project aims not only to create a robust, extensible, open source code base that covers the major common components required to build an SDN solution, but also to have a thriving and growing technical community contributing to the code base, using the code in commercial products, and adding value above, below and around[1].

1.2 Scope

While searching for information about the OpenDaylight Project, (explicitly in relation to OpenDaylight application development), it is easy to appreciate the large amount of information provided. Everything is on the Internet. However, as it is a new and a complex concept, it becomes quite difficult to obtain a comprehensive view on its functionality.

The main objective of this thesis is to bring the main points of information regarding OpenDaylight together. It specifically clarifies how an OpenDaylight application can be developed, and also provides examples and sources where the information is located.

1.3 Outline

- **Chapter 2** provides the relevant background information to this thesis. It introduces the reader into a number of various concepts including, Cloud Computing, Software Defined Networking and Network Function Virtualization.

- **Chapter 3** describes the OpenStack open source software, as it is a platform that creates private and public clouds (cloud computing deployment models). It also provides the steps to install a multi-node OpenStack deployment, as it is required when OpenStack Neutron uses OpenDaylight.

- **Chapter 4** analyzes the Software Defined Networking concept in depth. It also defines the OpenFlow protocol and the network emulator Mininet. Finally, this chapter describes and compares the different types of controllers to express the reasons why OpenDaylight was chosen for this thesis as opposed to other existing controllers.

- **Chapter 5** defines OpenDaylight. It presents an overview of the current projects, and illustrates and describes the technical aspects. Ultimately, it defines the OpenDaylight Controller project. Here is where important concepts such as AD-SAL and MD-SAL appear.
• **Chapter 6**, one of the core chapters of this thesis, explains step-by-step how to develop an OpenDaylight application. It provides different application examples and the obtained results, proving that the OpenDaylight controller is effectively able to control a network.

• **Chapter 7**, the final chapter, the thesis conclusion, summarizes the accomplishments and results obtained. Also included is a brief discussion on possible future work to further pursue the subject researched in this thesis.
Chapter 2

State of the Art Technologies

This chapter presents a general overview of the 5th Generation Networks. This includes Virtualization, Cloud Computing and its services and deployment models. Also in this chapter is a brief introduction to Software Defined Networking, Network Function Virtualization and the differences between them.

2.1 Cloud Computing

Before starting with the definition of cloud computing, a short explanation of the concept of virtualization is necessary. Virtualization is a technique of creating a virtual version of a device or a resource such as a server, a storage device, a network or even an operating system where the resource is divided into one or more execution environments.

In general terms, virtualization represents a physical server partitioned into multiple virtual servers. Each virtual machine (VM) can interact independently with other devices, applications, data and users, as if it was a separate physical resource. Different virtual machines can run different operating systems and multiple applications simultaneously using a single physical computer. The software that enables virtualization is called Hypervisor. This software, also known as 'virtualization administrator', is located between the hardware and operating system, separating the operating system and hardware applications.
In short, virtualization is no more than the creation of a virtual version of any technology re-
source through software. This is the technology used by cloud computing.

Cloud computing could be defined as a technology concept and a business model which pro-
vides storage services, access to computing resources and essentially, the use of them over the
Internet. Basically, cloud computing is the term used for the delivery of hosted services over the
"cloud". This term is used as a metaphor for Internet network diagrams of the infrastructure that
it represents.

Cloud computing is a paradigm in which information is stored permanently on Internet servers
and sent to temporary caches of the customer, including desktops, entertainment centers, lap-
tops, etc[2]. With cloud computing it is possible to increase capacity or add capabilities without
investing in a new infrastructure, new employees, or paying for the license of a software. Cloud
computing covers any subscription-based service or payment for use in real time over the Inter-
net.

Since cloud computing appeared the number of network-based services have increased. This
brings benefits to both providers and users. The providers can offer services faster and more
efficiently, and the users who are able to access them, can enjoy the transparency and immedi-
acy of the system. Likewise, the consumer saves labor costs or costs in economic investment
(local, specialized equipment, etc.). The consumer pays for use and as needed, elastic scale up
and down in capacity and functionalities.

A simple example of cloud computing would be Google Docs / Google Apps, a system of
electronic documents and applications. For its use software needs to be installed or have a
server. The only requirement needed to use any of their services is an Internet connection.

Cloud computing presents the following key features:

- **Agility**: improving capacity to provide technological resources to the user by the supplier.

- **Cost**: cloud computing providers argue that costs are reduced. A public provision model
  in the cloud becomes capital expenditures in operating expenses. This reduces entry bar-
  riers, as infrastructure is typically provided by a third party and not acquired by one-time
  or infrequent intensive computing tasks.

- **Scalability and elasticity**: resource procurement over self-service basis in nearly real
time, without loads of high durability.
• Device and location independence allows users to access the system using a web browser, regardless of the location or device used (e.g., PC, mobile phone).

• Virtualization technology enables sharing servers and storage devices and greater use. Applications can be easily migrated from one physical server to another.

• **Performance**: cloud systems control and optimize the use of resources automatically; this feature allows the monitoring, the control and the reporting of it. This capability provides transparency for both the consumer and the service provider.

• **Security**: this may improve due to the centralization of data. Security is often as good or even better than traditional systems, partly because providers are able to apply resources to the solution of security issues.

• Simpler **maintenance** of cloud computing applications; each application does not need to be installed on each user’s computer and can be accessed from different locations.

### 2.1.1 Cloud services

There are three basic types of services that constitute the business model of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The path taken to define these terms starts with the service most related to hardware, IaaS, until reaching the most relevant to software, SaaS, through the one which enables the development tools, PaaS.

**Infrastructure as a Service (IaaS)**

This service offers customers storage space or processing capacity on their servers. Hence, users will have a "hard drive of unlimited capacity" for their use and a processor with an almost infinite performance, just restricted to the type of service contracted. Basically it means having access to hardware located over the Internet.

Some examples would be Amazon EC2, Windows Azure, Rackspace, Google Compute Engine.

**Platform as a Service (PaaS)**

The Platform Service provides the IT development tools, so that users can build their application or software pieces without purchasing and implementing such tools in their local computers. The Platform Service provides the IT development tools, so that users can build their application or software pieces without purchasing and implementing in their local computers such tools. This service has two clear advantages for the application developer: there is no need to purchase the expensive license to develop the market tools. On the other hand the service provider ensures
Chapter 2. *State of the Art Technologies*

that these tools are in optimal maintenance situation.

For example, AppScale allows a user to deploy some applications written for Google App Engine to their own servers, providing datastore access from a standard SQL or NoSQL database.

**Software as a service (SaaS)**

SaaS is a distribution model which gives customers access to software applications over the Internet. This is often called Web services. Thanks to these services, application maintenance, and technical support operations are not necessary. These Web services can reach any kind of company regardless of their size or geographical location. This is a model that links the product (software) to the service, to provide companies a complete solution to optimize their costs and resources.

Google Apps, Twitter, Facebook and Flickr are all examples of SaaS, with users able to access the services via any internet enabled device.
2.1.1 Examples

Several large companies are the ones dedicated to procure these services, promoting an easy access to the information, low cost, and scalability. We can mention many features that make us think of the comfort they give us:

- Google Apps offers the service of business applications such as Gmail, Google Talk, Google Calendar and Google Docs, etc.
- Amazon Web Services: the services offered are the Amazon EC2™, Amazon S3™, SimpleDB™, Amazon SQS™.
- Microsoft Azure procures operating system services, hosting and systems development.

2.1.2 Deployment Models

Public clouds

The cloud infrastructure is available to the general public and belongs to the organization which sells cloud computing services. Services are shared without an exhaustive control over the location of the information that resides on the provider’s servers. However, public and unsafe are not synonymous. For the user of these services are all operating costs (OPEX).

The following are the main characteristics in this form of implementation:

- Public clouds reduce timeframe for service availability.
- Performing monetary investment for implementation is not required.
- It allows the outsourcing of all basic functions of a company to a cloud service provider.
- It encourages the use of standard software packages.
- Initial fees are lower than other implementations. Costs of public cloud are variable, fulfilling the principle of pay per use.
- Corporate data is hosted in the public cloud with the rest of provider customers, which means, aside from not being able to physically locate such information, some requirements of high demands on safety and protection data are absolutely necessary.
Private clouds

This model is for customers that need a platform and applications for their own exclusive use, due to the importance of the managed information. The cloud infrastructure is part of one organization, which provides this infrastructure as a service to their departments. It can be managed by the organization or by a third party and may exist within the same institution or out of it. It solves some of the problems of the public clouds (security, control data, services, etc.), but it involves capital expenditures (CAPEX).

Private clouds are a good choice for companies that require high data protection and have service issues. In private clouds the client controls which applications to use and how to use them. The company owns the infrastructure and decides which users are allowed to use it. The main characteristics are listened as follows:

- Short period of time for service availability and high flexibility in the allocation of resources.
- Unlike public clouds, economic investment is required for implementation of the solution selected.
- This model brings associated systems and local databases.
- It offers the ability to leverage existing staff and investments in information systems previously made.
- The company has total control over the administrated infrastructure, systems and corporate information.
- There is better control and supervision of the security requirements and protection of the stored information.

Hybrid clouds

Here, the private infrastructure is augmented with cloud computing services of public infrastructure. This allows a company to maintain control of its main applications and leverage cloud computing published only when necessary.

This model combines characteristics of public and private clouds, thus part of the service can be offered privately (infrastructure) and another part on a shared basis (development tools):
• This model offers greater flexibility in providing IT services, while greater control over business services and data are maintained.

• With a hybrid cloud solution, as well as the models described above, the timeframe for service availability is reduced.

• It involves higher complexity of integrating the cloud solution, as a result of the combination of two different cloud services implementations.

• This model allows the integration of the best features of the cloud models, regarding the data control and the basic functions of the entity management.

Community clouds

This service is used by different cloud organizations whose functions and services are common, thereby enabling collaboration among stakeholders.

Examples of this form of implementation are the healthcare community clouds which facilitate the access to applications and critical information on general health issues.

In principle, its strengths and weaknesses are located between the private and the public. In general, the set of available resources with community cloud is higher than in the private, with obvious the benefits that it entails in terms of elasticity. However, the amount of resources is lower than the ones of the public cloud, limiting the elasticity regarding public cloud. Moreover, the number of users of this type of cloud is lower than the public cloud, which gives it greater benefits in security and privacy.

2.2 Software Defined Networking

With Software Defined Networking (SDN) the global network control is achieved by centralizing the logic function control plane, and network-operations organizations can manage with a group of network devices as a single entity. With SDN, network flows are controlled in an abstraction level of the global network, rather than at the level of individual devices, usually, but not always, with the help of the OpenFlow protocol.

The group that is most involved with the development of standards based on SDN is the ONF[3] (Open Networking Foundation). The ONF was launched in 2011 and its vision is to make the SDN based in OpenFlow as the new standard for networks. In order to achieve this vision,
the ONF took on the responsibility of leading the standardization of OpenFlow protocol. The amplitude of the ecosystem SDN is reflected in the fact that the ONF currently has over 140 members of various types, including suppliers providing silicon and switches, network devices, controllers, test equipment, telecommunication services, data center services hyper-scale and smartphones.

A layered architecture for SDN is shown in the figure above. In this architecture, the control plane functionality is centralized in the driver software SDN.

In the following chapter the SDN definition is completed in more detail.

### 2.3 Network Functions Virtualization

The manufacturers of telecommunication equipment traditionally developed their products (base stations, firewalls, softswitches, routers, etc.) on proprietary hardware, optimized for each type of application. These devices were not capable of withstanding other functions efficiently in comparison to those that had been considered in their initial design. Although currently proprietary platforms are the most effective for certain applications; many nodes network, or at least its control plane are supported more efficiently on standard or COTS(Commercial Off-The-Shelf).
The convergence of telecommunications networks to the world "all IP" can take advantage of the huge progress that has been in recent years in virtualization technologies for large data centers. Network Functions Virtualization (NFV) decouples services and applications to a specific hardware in order to provide a more elastic model, where resources can be used when and where necessary. Thus NFV allows consolidate different types of proprietary network infrastructure in a large volume of standard hardware, which implies for operators large cost savings, simplification of the introduction of new services and more innovation.

2.3.1 NFV technology

Before going into more details about NFV, it is necessary to clarify that NFV and SDN are different technologies. As both are linked to the "Cloud" concept in the telecommunications world, they are often confused or wrongly used. Similar to NFV, SDN (Software Defined Networking) involves a transformation in the traditional architecture of telecommunication network operators. In SDN architecture network intelligence is centralized and provides a separation between control plane and data. This separation helps network virtualization. Thus, NFV is highly complementary to SDN, allowing to take a maximum benefit of cloud computing. Both are very beneficial for operators in term of cost and flexibility, but they do not depend strictly on each other. NFV can be virtualized and deployed without SDN and vice versa. NFV can be applied to both the control plane and the data plane, and to both fixed and mobile networks, even though the data plane is which represents higher challenges.

2.3.2 NFV definition

Network Functions Virtualization (NFV) is an initiative to virtualize network services that are now performed by proprietary hardware. This is intended to decrease the amount of proprietary hardware needed to implement and operate network services.

NFV aims to dissociate the functions of network devices proprietary hardware and to enable network services which are now performed by routers, firewalls, load balancers and other proprietary devices which are hosted on virtual machines (VM) and network functions that are under the control of a hypervisor. As server capacity can be added via software, there will be no longer a need for network administrators to supply their data centers, which will reduce both capital expenditures (CAPEX) and operational costs (OPEX).

For instance, if an application is running on a VM and requires more bandwidth, the administrator can move the virtual machine to another physical server or supply another VM on the
original server to take part of the load. Having this flexibility will allow IT department to respond quicker to changing business goals and demands of network services.

![Classical Network Appliance Approach vs Network Functions Virtualisation Approach]

**FIGURE 2.3: NFV vision**
Source: https://network-functions-virtualization.com

### 2.3.2.1 European Telecommunications Standards Institute

The standardization of NFV is being led by the ETSI[4] (European Telecommunications Standards Institute). After several informal conversations and meetings since April 2012, the NFV ETSI ISG (Industry Specification Group) was formally created in November 2012, by seven telecommunications operators (ATT, BT, Deutsche Telekom, Orange, Telecom Italia, Telefónica and Verizon). The NFV ISG is open to both organization members and the ETSI. Today consists of over 150 companies, including main telecommunications operators and telecommunications infrastructure and IT suppliers.

Although ETSI is an organization for development standards, the objective of the NFV ISG is not to produce standards, but to get an industrial consensus on the technical and business requirements of the NFV, and also to agree on common approaches to meet these requirements under
a common architecture. Results are published openly and shared with key standard groups and industry forums such as IETF, NGMN Alliance, ONF, TM Forum etc.; in order to get a broader collaborative effort. The NFV ISG collaborates with other standardization bodies in case any kind of standardization is necessary to achieve the requirements.

The first specifications[5] of the ETSI ISG were published in October 2013: GS NFV 001 "Use Cases", GS NFV 002 "Architectural Framework", GS NFV 003 "Terminology for Main Concepts in NFV", GS NFV 004 "Virtualisation Requirements" y GS NFV-PER 002 "Proofs of Concepts Framework”. These five documents establish a framework of fundamental work for NFV development, covering: use cases, requirements, architecture framework and terminology. In addition a framework is established to coordinate and promote public demonstrations of the key aspects of NFV. This will make it easier for network operators and solution providers of the NFV to work together. This subsequently improves the interoperability and the development of an open ecosystem which will facilitate global scale economies.

GS NFV-REL 001 Ver 1.1.1 "Resiliency Requirements" and GS NFV-INF 001 Ver 1.1.1 "Infrastructure Overview" are the last published specifications on January 2015, the following table is a list of recently published specifications on Network Functions Virtualisation:
Currently ETSI ISG is working on several new specifications, meant to be published throughout this year; the fields of these specifications are as follows:

- Trust
- Ecosystem
- Management and Orchestration
- Assurance
- Virtualization Technologies

**Table 2.1: Published specifications on NFV**

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GS NFV-INF 001</td>
<td>Network Functions Virtualisation (NFV); Infrastructure Overview</td>
</tr>
<tr>
<td>GS NFV-INF 004</td>
<td>Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain</td>
</tr>
<tr>
<td>GS NFV-REL 001</td>
<td>Network Functions Virtualisation (NFV); Resiliency Requirements</td>
</tr>
<tr>
<td>GS NFV 002</td>
<td>Network Functions Virtualisation (NFV); Architectural Framework</td>
</tr>
<tr>
<td>GS NFV 003</td>
<td>Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV</td>
</tr>
<tr>
<td>GS NFV-INF 003</td>
<td>Network Functions Virtualisation (NFV); Infrastructure; Compute Domain</td>
</tr>
<tr>
<td>GS NFV-INF 005</td>
<td>Network Functions Virtualisation (NFV); Service Quality Metrics</td>
</tr>
<tr>
<td>GS NFV-MAN 001</td>
<td>Network Functions Virtualisation (NFV); Management and Orchestration</td>
</tr>
<tr>
<td>GS NFV-SWA 001</td>
<td>Network Functions Virtualisation (NFV); Virtual Network Functions Architecture</td>
</tr>
<tr>
<td>GS NFV-SEC 003</td>
<td>Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance</td>
</tr>
<tr>
<td>GS NFV-PER 001</td>
<td>Network Functions Virtualisation (NFV); NFV Performance and Portability Best Practices</td>
</tr>
<tr>
<td>GS NFV-PER 002</td>
<td>Network Functions Virtualisation (NFV); Proofs of Concept; Framework</td>
</tr>
<tr>
<td>GS NFV-INF 007</td>
<td>Network Functions Virtualisation (NFV); Infrastructure; Methodology to describe Interfaces and Abstractions</td>
</tr>
<tr>
<td>GS NFV-SEC 001</td>
<td>Network Functions Virtualisation (NFV); NFV Security; Problem Statement</td>
</tr>
<tr>
<td>GS NFV 001</td>
<td>Network Functions Virtualisation (NFV); Use Cases</td>
</tr>
<tr>
<td>GS NFV 004</td>
<td>Network Functions Virtualisation (NFV); Virtualisation Requirements</td>
</tr>
</tbody>
</table>
• Privacy and Regulation
• Pre-deployment Testing
• Acceleration Technologies
• Security Guide
• Security Monitoring
• Testing Methodology

2.3.2.2 Benefits of NFV

NFV will radically transform the way network operators design and dimension their networks, enabling the consolidation of network services on servers, hubs and standard storage systems, which can be located in data centers, network nodes or in the customer premises. NFV can provide significant benefits to network operators and their customers, which include:

• **Reduce costs**: CAPEX and OPEX are reduced by decreasing equipment costs, energy consumption, logistics, operations, etc. The largest volume of purchases of the same infrastructure standard allows better discounts. Cycles of hardware introduction improvements in processing and memory are reduced as well. This also facilitates operation and maintenance through simplification and automation, reducing the need of training and specialized personnel. NFV allows a more efficient use of hardware resources and software licenses in each of the potential demanding situations for network resources (users, bandwidth, etc.). This improves efficiency, robustness and scalability of the network and drastically reduces the time dedicated to planning and dimensioning of network resources.

• **Simplify the deployment of new services**: The introduction of new services in the network of operators is very slow and costly, often facing the increasing number and complexity equipment in their networks. This represents a large certification process with specialized employees in each one of these services. NFV accelerates the deployment of new services, giving a faster return on investment. Sharing resources between services, applications and users, provides better resource utilization and more flexibility to scale to both larger and smaller capacities. New services can be introduced in a more controlled-way, with less risk and cost. If the service is not commercially successful or if customers suffer significant incidents, it will be able to finish quickly with the service of the network, without incurring significant losses for the purchase of infrastructure.
• **Promoting innovation**: NFV accelerates innovation and service differentiation; it favours an open ecosystem, since the absence of proprietary hardware reduces the entry barriers for new providers of software. It is also easier to input more suppliers, as the scale economies needed to cover investments in features based in hardware are never more applicable the development of software based functions.
Chapter 3

OpenStack

Openstack is a project for building and managing cloud computing platforms (public and private clouds) to provide an infrastructure as a service (IaaS). This project aims to be an open-source software easy to implement and massively scalable.

3.1 OpenStack Objective

"Create a free software platform for cloud computing that meets the needs of providers of public and private clouds, regardless of size, easy to implement and massively scalable.[6]"

OpenStack is a collection of Open Source technologies that provide software for scalable deployment of cloud computing. OpenStack provides Infrastructure as a Service (IaaS) and is a project initiated in 2010 by the company Rackspace Cloud and the US space agency, NASA. Currently more than 150 companies have joined the project such as AMD, Intel, Canonical, SUSE Linux, Red Hat, IBM, Dell, HP, Cisco, etc. OpenStack is free software under the terms of the Apache license.

OpenStack can be used by any organization looking to deploy a large-scale cloud for both private and public use. OpenStack is an interesting project for almost any type of organization: small and medium enterprises, administration, large corporations, service providers, value-added enterprises, data centers, etc.
3.2 What is OpenStack?

Basically, it is an Open Source software used to build public and private clouds. OpenStack represents both a community and a Free Software project as software to help organizations to execute their own clouds for computing or virtual storage.

From the point of view of software, OpenStack is a free software collection projects maintained by the community which include several components. Through these services, OpenStack provides a complete operating platform for administration and management of clouds. The most important components of OpenStack are explained in more detail:

**Keystone (Identity)** provides authentication and authorization services for all OpenStack, and also a catalog of these services in a particular cloud.

**Services:**

- **Identity**: Provides validation of authentication credentials of users, "tenants" and roles.
- **Token**: validates and manages tokens used for authentication once credentials have been verified.
- **Catalog**: provides information about available endpoint. With which you can access to services for each of the tenants.
- **Policy**: provides engine based authorization rules.

**Glance (Image)** provides a catalog and a repository for images.

**Services:**

- **glance-api**: is the main interface that provides access to Glance APIs. It addresses customer requests for service glance-registry, manages the images in the back-end and in the cache where these are stored.
- **glance-registry**: manages information of images (size, type, etc.) into the database.

**Cinder (Block Storage)** provides persistent storage for VMs hosted on the cloud.

**Services:**

- **cinder-api**: provides authentication and route requests to the other cinder services. It has the API to handle the volumes and their interaction with virtual machines and their backups.
• **cinder-volume**: manages the block storage devices, specifically which are defined as back-end.

• **cinder-scheduler**: locates service requests to the appropriate back-end storage.

**Neutron (Network)** provides virtual networks as a service between devices managed by other OpenStack services, such as a virtual machine Nova. It allows users to create their own networks and then link them to the devices they want.

Services:

• **quantum-server**: accepts and routes requests to the appropriate plugins for processing.

• **quantum-plug-in and quantum-agent**: connect and disconnect ports, create networks, sub-nets, provide IP addressing. The plugins depend on the provider and the technology that OpenStack supports and wants to use.

**Nova (Compute)** recovers images and associated metadata, and transforms the requests of users in virtual machines.

API Services:

• **nova-api**: accepts and responds to the API calls by customers. It supports OpenStack APIs, Amazon EC2 and special API for management, as well as orchestrating many activities: running instance and enforce certain policies.

• **nova-api-metadata**: accepts requests from virtual machine instances. It is generally used when nova-network is used as multi-host mode.

"Compute" Services:

• **nova-compute**: creates and terminates virtual machines through hypervisor APIs and updates the information in the database.

• **nova-scheduler**: makes a request to create virtual machine instance and determines which hypervisor will run.

• **nova-conductor**: makes a request to create virtual machine instance and determines which hypervisor will run.

Console Services:
• nova-consoleauth: authorizes tokens for users which access to consoles.

• nova-novncproxy: provides a proxy to access consoles instances via VNC. It supports novnc browser-based clients.

• nova-xvpnovncproxy: the same as novncproxy but this one only supports java client specifically designed for OpenStack.

**Swift (Object Store)** provides object storage. This is not a file system, it’s more like a container that can store files and then recover.

Services:

• Swift-proxy-server: accepts requests from the API and HTTP requests to upload, download and create containers.

• Account: manages the defined accounts.

• Container: manages the folder relationship within the service.

• Objects: manages the files (objects) in the storage nodes.

![Horizon Dashboard](image.png)

**Figure 3.1:** Horizon Dashboard
Horizon (dashboard) is the graphical interface via Web (see Figure 3.1) through which is possible to handle much of the functionality of OpenStack via APIs (throw instances, assign IP, create user, etc.). An example would be:

The table above shows the components and their description:

<table>
<thead>
<tr>
<th>Service</th>
<th>Project Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dashboard</td>
<td>Horizon</td>
<td>Computational resources on demand management</td>
</tr>
<tr>
<td>Compute</td>
<td>Nova</td>
<td>Computational resources on demand management</td>
</tr>
<tr>
<td>Networking</td>
<td>Neutron</td>
<td>Networking automation</td>
</tr>
<tr>
<td>Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Storage</td>
<td>Swift</td>
<td>Storing objects in distributed environment</td>
</tr>
<tr>
<td>Block Storage</td>
<td>Cinder</td>
<td>Management volumes to storage blocks (IBM, EMC, HP, Red Hat / Gluster, Ceph / RBD, NetApp, SolidFire and Mexenta)</td>
</tr>
<tr>
<td>Shared Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identity Service</td>
<td>Keystone</td>
<td>Control images of O.S</td>
</tr>
<tr>
<td>Image Service</td>
<td>Glance</td>
<td>Authentication and authorization control</td>
</tr>
<tr>
<td>Telemetry</td>
<td>Ceilometer</td>
<td>Measurement and monitoring features</td>
</tr>
<tr>
<td>Higher-level services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orchestration</td>
<td>Heat</td>
<td>Service orchestration</td>
</tr>
<tr>
<td>Database Service</td>
<td>Trove</td>
<td>Database as a service</td>
</tr>
</tbody>
</table>

TABLE 3.1: OpenStack Components
3.3 How to install multi nodes in OpenStack

OpenDaylight brings together a number of virtual networking approaches; one of them is the integration approaches with OpenStack Neutron[7]. OpenDaylight has a driver for Neutron ML2 (Modular Layer 2) plugin to enable communication between them. OpenDaylight manages the network flows for the OpenStack compute nodes via the OVSDB south-bound plug-in.

This section focuses to discuss how to install multi nodes in OpenStack (minimum requirement) to configuring Neutron to use OpenDaylight[8].

The reference deployment used in this section is a 3 node cluster:

- **Compute Node** - two network interfaces: eth0 (network connectivity) and eth1 (internal network for Management use)
- **Control Node** - two network interfaces: eth0 (network connectivity) and eth1 (internal network for Management use)
- **Network** – three Network interfaces: eth0 (network connectivity), eth1 (internat network for Management use) and eth2 (data network for connectivity between Virtual Machines)

**Preparation of Control Node**

Two network interfaces :

```bash
$ vi /etc/network/interfaces

#External Network

auto eth0
   iface eth0 inet static
      address 192.168.100.10
      netmask 255.255.255.0

gateway 192.168.100.1

#Management Network

auto eth1
   iface eth1 inet static
      address 10.0.0.10
      netmask 255.255.255.0

gateway 10.0.0.1

$ service networking restart

#set the hostname
```
Chapter 3. OpenStack

$ hostname controller.flux7.com
$ echo controller.flux7.com &gt; /etc/hostname

Add OpenStack Ubuntu Cloud Archive repository

Adding Havanna release repository:

$ apt-get install python-software-properties
$ add-apt-repository cloud-archive:havana
$ apt-get update
$ apt-get dist-upgrade (Optional)

Install MySQL server

MySQL as database back-end:

$ apt-get install python-mysqldb mysql-server
# /etc/mysql/my.cnf
#bind-address 192.168.100.10
(Comment out this line or change 192.168.100.10 to 0.0.0.0)

Install Messaging server

RabbitMQ as messaging/broker service:

$ apt-get install rabbitmq-server
$ rabbitmqctl change_password guest &lt;New_Pass&gt;

Preparation of Compute Node

Two network interfaces:

$ vi /etc/network/interfaces
#Management Network

    auto eth0
    iface eth0 inet static
    address 10.0.0.20
    netmask 255.255.255.0

gateway 10.0.0.1

#Data Network

    auto eth1
    iface eth1 inet static
    address 11.0.0.20
    netmask 255.255.255.0
Add OpenStack Ubuntu Cloud Archive repository

Add Havanna release repository:

$ apt-get install python-software-properties
$ add-apt-repository cloud-archive:havana
$ apt-get update
$ apt-get dist-upgrade (Optional)

Install MySQL python library

MySQL client and MySQL Python library:

$ apt-get install python-mysqldb

Preparation of Network Node

Three network Interfaces:

$ vi /etc/network/interfaces
#External Network
auto eth0
  iface eth0 inet static
  address 192.168.100.30
  netmask 255.255.255.0

gateway 192.168.100.1

#Management Network
auto eth1
  iface eth1 inet static
  address 10.0.0.30
  netmask 255.255.255.0

gateway 10.0.0.1

#Data Network
auto eth2
  iface eth2 inet static
address 11.0.0.10
netmask 255.255.255.0

gateway 11.0.0.1

$ service networking restart

# Set the hostname

$ hostname network.flux7.com
$ echo network.flux7.com &gt; /etc/hostname

---

**Add OpenStack Ubuntu Cloud Archive repository**

**Add Havanna release repository:**

- $ apt-get install python-software-properties
- $ add-apt-repository cloud-archive:havana.com
- $ apt-get update
- $ apt-get dist-upgrade (Optional)

---

**Install MySQL python library**

**MySQL client and MySQL Python library:**

- $ apt-get install python-mysqldb
Chapter 4

Software Defined Networking

This chapter is performed a brief introduction to the Software-Defined Networking (SDN) concept and a detailed study of OpenFlow technology. OpenFlow is a new technology based on SDN. This concept was defined to break with the conventional model of network where the switch is the one that decides the actions to do. OpenFlow technology moves the control logic to an external controller (server). This controller is responsible for deciding the actions that the switch must perform. This communication between the controller and the Data Path is made, on the network itself, using the protocol provided by OpenFlow.

4.1 SDN, an Introduction

SDN is a revolutionary way of managing a network. Its main feature is to separate the management from the switching. This new architecture aims to decouple the network infrastructure from the applications and services it provides. The network will work as a logical entity, separate and independent of the applications and services previously mentioned.

By way of explanation, the administrator is completely able to define and manage the behavior of the network. As the OFN (Open Networking Foundation) defines, SDN is a dynamic, manageable, cost-effective, and adaptable architecture, which protects existing investments while enabling adaptation to future needs.

This is achieved by centralizing the control of the network. An element called Controller implements the behavior rules of the switching elements, based on rules previously defined by the network administrator.
4.1.1 Motivation to use SDN

Today, networker users are becoming extremely demanding and exigent. It is becoming more complicated every day for the telecommunication industry to meet these demands due to the huge use of mobile devices.

The current architecture is composed of the Data plane and the Control Plane. The Data plane is in charge of looking at a routing table from which it can read where to forward the packets; the Control plane acts as the brain of the network element – it manages the network, and the system configuration - deciding where traffic is sent. For instance, if a network consists of 10,000 network devices, 10,000 control planes need to be administrated and keep all of them updated.

With SDN the control plane is not implemented on the network element, but on a centralized server. Applications have direct access to the control plane. See Figure 3.1.

The main benefits of using SDN are the reduction of costs in equipment and network operations, the increased speed of innovation, the ability to maximize an open ecosystem and the accelerated introduction of new services and scalability. In addition, SDN creates new business opportunities for customers by providing access to information on the status of the network and by allowing them to manage traffic flows in a smarter and efficient manner.
Chapter 4. Software Defined Networking

The current network can evolve into a deploying application and service platform is able to respond quickly to changing user, business or market needs. With SDN application it is possible to demand resources in real time.

4.1.2 Architecture

According to the Open Networking Foundation, the SDN architecture is [9]:

- **Directly programmable**: Network control is directly programmable because it is decoupled from forwarding functions.
- **Agile**: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- **Centrally managed**: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- **Programmatically configured**: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- **Open standards-based and vendor-neutral**: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

As mentioned before, in SDN networks the main fact is the separation between control plane and data plane. Hence, we could see it as a set of transmission elements connected by switching elements, the same as in a traditional layer, but adding the control layer above it, where the Controller manages the network.

More concretly, a SDN network is divided into three logical layers, as shown in the Figure 4.3.

The bottom layer consists of the network infrastructure, switches/routers, hosts and transmission media. The interlayer is composed of the SDN Controller, which has a global view of the network and incorporates the Network Operating System (NOS). The NOS is the decision maker and also programs the flow tables of the elements of the bottom layer. Using the SouthBound
APIs, the Controller can communicate with the switching elements of the network. These APIs enable the SDN Controller to dynamically make changes according to real-time demands and needs. OpenFlow is the first and probably the most well-known southbound interface, which will be explained later.

The top or initial layer is where the philosophy in relation to traditional networks completely changes. It is called the application layer. This layer is composed of user applications, where are the NorthBound APIs. The function of these NorthBound APIs is to communicate usage patterns to the control layer, where decisions are made. In short, applications define how the network is going to be used, then they communicate it to the SDN Controller, which makes the appropriate decisions and informs the lower layer using the SouthBound APIs.

The main advantages of SDN over traditional networks are: testing and deploying new applications in real network, capital and operating expenses are minimized and there is a centralized management of each switch.

### 4.2 Controllers

A SDN controller, basically provides an interface to the device layer. The network management logic encamped in the applications are executed via this interface, which may be translated
as implementation of high level policies over the device layer. A SDN controller is generally described as a software system, or a collection of systems. The features that it offers are:

- Management of the state of the network.
- A data model that captures high-level connections between the managed resources, policies and other services provided by the controller.
- A mechanism of device discovery, topology and service; a path calculation system and other information services.
- A secure session control over TCP (Transmission Control Protocol) between the controller and associated agents in the network elements.
- A standards-based protocol (OpenFlow).
- A set of APIs, often RESTful (Representational State Transfer) that expose controller services to management applications.

Therefore, in SDN, the central controller is the one that guides the overall network performance based on the application requirements. Some examples [10] of existing open source controllers are: NOX, POX, Beacon, Floodlight, Ryu, Trema and OpenDayLight (ODL).

The following table lists compares some of the mentioned controllers.

<table>
<thead>
<tr>
<th>Controllers</th>
<th>NOX</th>
<th>POX</th>
<th>BEACON</th>
<th>Floodlight</th>
<th>ODL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>C++</td>
<td>Python</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>OpenStack</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GUI</td>
<td>Python+QT4</td>
<td>Python+QT4</td>
<td>Web</td>
<td>Web</td>
<td>Web</td>
</tr>
<tr>
<td>Documented</td>
<td>Medium</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>O.S</td>
<td>Linux</td>
<td>Linux, Mac OS Windows</td>
<td>Linux, Mac OS Windows</td>
<td>Linux, Mac OS Windows</td>
<td>Linux, Mac OS Windows</td>
</tr>
<tr>
<td>OpenFlow</td>
<td>OF v1.0</td>
<td>OF v1.0</td>
<td>OF v1.0</td>
<td>OF v1.0, OF v1.3</td>
<td>OF v1.0, OF v1.3</td>
</tr>
<tr>
<td>Installation</td>
<td>Difficult</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>REST API</td>
<td>NO</td>
<td>Basic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Abstraction layer above SB protocols</td>
<td>NO</td>
<td>NO</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Table 4.1: Controllers comparison**
4.2.1 NOX

Everything began with NOX [11]. This was the first accepted Openflow controller by the researcher’s community whose great interest led to their work around it. However, since 2010, it stopped being an important controller and there was subsequently no more documentation and updates available.

The language used in NOX is C++ and it is known that the environment provided by this controller is tedious, and difficult to program. This controller is shipped with a Python+QT based GUI(Graphical User Interface). A missing feature is that it does not support topologies with loops.

4.2.2 POX

This is often called the younger sibling of NOX, as it was both created and maintained by the same organization as NOX. POX was created in an attempt to create a controller with a better developer environment than NOX.

POX [12] is written in Python and, in addition to supporting the NOX GUI, POX also provides a web-based GUI.

4.2.3 Beacon

The code for this controller is written in Java, in a simple and expressive style. The developer only needs to program in an IDE (Integrated Development Environment), Eclipse for example, which is in charge of the compilation and library issues. It was also the first controller to be managed in a built-in web user interface. The main issue that Beacon presents is that it is not able to handle topologies with loops or devices with multiple attachment points, it only supports start topologies [13].

4.2.4 Floodlight

The Floodlight [14] controller is Java-based OpenFlow Controller. Originally developed by David Erickson at Stanford, the Floodlight controller derived from the Beacon controller. However, unlike Beacon, Floodlight supports topologies with loops and non-OpenFlow domains, and multiple device attachment points. Along with these features, Floodlight is built with Apache Ant, whereby it is possible to export easily JARs (Java ARchive) and replaces OSGi [15] (Open
Services Gateway Initiative) with a custom, Java-based, simple module loading system.

There is also a really supportive and effective community. Its functionalities are exposed through a REST API (Representational State Transfer) and it has a web-based UI (User Interface) and a Java-based GUI, called Avior. It is important to also mention that it can also be run as network backend for OpenStack using a Neutron plug-in.

### 4.2.5 OpenDaylight

OpenDaylight (ODL) is an open source project under linux distribution. This controller, as Beacon and Floodlight, is also based on the OSGi architecture. It supports network programmability via southbound protocols, a bunch of programmable network services, a collection of northbound APIs, very easy to use and intuitive, and a set of applications.

There is no doubt that ODL is the most successful open-source SDN controller with a very large community. This is largely due to the successful collaborations between member companies and excellent commitment from the existing members, such as Arista Networks, Big Switch Networks, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, Red Hat and VMware. OpenDaylight is also the most documented controller, with numerous mediums such as wiki, IRC, listservs, meetup groups, hackfests, summits, and video tutorials.

As it will be explained in section 5.3, there are three available methods to write an application using ODL controller. Then, questions such as how it is possible to communicate to the plugin in three different ways or what happens if the devices are using a different protocol. Thanks to the Service Abstraction Layer (see section 5.3.1). This layer translates the required services and orders, making OpenDaylight controller more flexibility.

For these reasons, OpenDaylight is the controller that will be used to complete this project.

### 4.3 OpenFlow

There is considerable confusion with the OpenFlow term. Sometimes is refered as a synonymous of SDN. Other times is confused with a type of controller.

However, OpenFlow[16] is the protocol used by a SDN server. With this protocol a server can order the network switches where to send the packets. In traditional networks each switch has
proprietary software which tells the server what to do. However, with OpenFlow, the migration of packet decisions is centralized. Now the network can be programmed independently of the switches and data center.

Therefore, OpenFlow would have two main functions:

- Allowing users to define flows
- Deciding the path to take through the network

Before going into more detail about what OpenFlow is, talking about the OpenFlow Switch is necessary.

### 4.3.1 OpenFlow Switch

Switch OpenFlow is composed of flow tables and a channel towards the controller. It is divided in three parts:

- **Secure channel:** It is a channel needed to create the communication between the switch and the controller. This communication is possible thanks to the OpenFlow protocol.
- **OpenFlow Protocol:** Southbound API, a communication protocol between the control layer and the infrastructure layer.
- **Flow table:** indicates how the switch must process the arriving flows.
An OpenFlow switch may have different flow charts, where the searching of incoming packets match is performed. Each entry in a table flow consists of three fields:

- Header, which defines the flow
- Counters, which counts the matching packets. Updated per flow, per table, per port and per queue
- Actions to take when it finds a match between a package and a flow table

The processing of a packet by an OpenFlow Switch is as follows:

As previously mentioned, in traditional networks developing new applications and modifying the behavior of existing devices becomes more difficult every day. With SDN these problems disappear separating the control logic from devices to a centralized place: the SDN Controller.

OpenFlow protocol defines the protocol communication between the switch OpenFlow and the controller. The switch establishes communication with the controller in an IP address using a particular port. If the switch knows the IP switch controller, it initiates a standard TCP session.

When a connection is established, each side sends a HELLO message. Once the message is received, the receiver calculates the protocol version to use. If the version negotiated is supported, the connection will be initiated, otherwise it will send a HELLO-FAILED message and the connection will end.
4.3.2 SDN and OpenFlow

In traditional networks, the data plane and the control plane are tightly coupled on the same device. Therefore in traditional networks, the development of new applications and the modification of the behavior of existing devices is a very difficult task. SDN overcomes these problems by shifting the control logic from devices to the centralized place. The shifted control logic is called SDN Controller or Network Operating System (NOS). Thanks to the Controller, the functionality of the network is managed in a very efficient manner.

OpenFlow is a standard protocol that is used to provide a communication between the Controller and a dumb device. The Controller and dumb devices are called 'control plane' and 'data plane' respectively. The OpenFlow controller is responsible of deciding which action is to be performed by the switch. The decision approach is either Reactive or Proactive.

In the Reactive approach, when a packet arrives to the switch, the switch does not know how to handle that packet. Therefore the packet is sent to the controller. Controller is responsible of inserting a flow entry into the flow table of a switch using the Openflow protocol. The main disadvantage of this approach is that the switch is totally dependent of the controller decision. Thus, when a switch loses the connection with the Controller, it is not able to handle that packet. In the Proactive approach, the Controller fills the flow entries in the flow tables of each switch. This approach overcomes the limitation of Reactive approach because even if the switch loses
the connection with the Controller, it does not disrupt traffic.

4.4 Mininet

In order to experiment with SDN features, develop applications and encourage their use, a group of professors from Stanford decided to create Mininet.

Mininet is a network emulator. It creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command. In other words MiniNet is an OpenFlow testing platform. It is possible to access to each host individually, make a ssh connection and each host may individually run applications installed on the Linux system. Therefore, it is possible to simulate such complex custom scenarios through virtual components created using the Mininet Phyton API.

Mininet has a number of features that make it a suitable environment for development, testing, and teaching in the field of SDN:

- **Flexibility**: the use of programming languages and known operating systems allows the definition of new topologies and functionalities.

- **Interactivity**: the management and operation of the network must be done in real time, as if it was operating on a physical network.

- **Scalability**: real environments must allow scaling networks with a large number of switches on a single physical machine.

- **Realistic**: the developed prototype must match the real one as much as possible.

- **Share-ability**: a simple sharing of independent prototypes to test them in parallel.

Mininet is used in many areas such as research, development, learning, prototyping, testing, debugging and many other tasks.
The developed, and Mininet tested code can be exported to a real system without any changes. Mininet[7]:

- Provides a simple and inexpensive network testbed for developing OpenFlow applications
- Enables multiple concurrent developers to work independently on the same topology
- Supports system-level regression tests, which are repeatable and easily packaged
- Enables complex topology testing, without the need to wire up a physical network
- Includes a CLI (Command Line Interface) that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests
- Supports arbitrary custom topologies, and includes a basic set of parametrized topologies
- is usable out of the box without programming, but
- also Provides a straightforward and extensible Python API for network creation and experimentation

Mininet also possesses a large range of features that other virtualization systems do not:

- **Speed**: Implementing and testing a connectivity network takes seconds.
- **Larger scaling**: hundreds of hosts and switches.
- **Provides more bandwidth**: typically 2Gbps total bandwidth
- **Easy to install**

However, MiniNet has some drawbacks, among which includes the sharing of resources. As any virtualization resources the parent system is shared, and consequently the size and operation of the simulated network is conditioned by these resources. In addition, Mininet is not able to run OpenFlow switches or applications that are not compatible with Linux.

### 4.4.1 Procedure

As previously mentioned, Mininet provides a fast prototyping method to create, interact, customize and share software defined networking, as well as the possibility of being executed on real hardware. Furthermore, it is possible to combine virtualization, a CLI and an API.
Setting up

To create a network, Mininet emulates the different links, hosts, switches and controllers using different virtualization mechanisms of Linux operating system.

- **Links**: a virtual pair Ethernet acts as a network cable connecting two virtual interfaces. Thus, the packets sent through one interface are received by the other, each interface appearing as a fully functional Ethernet port for all types of systems and applications. The end of a virtual connection (pair) Ethernet can be connected to virtual switches, such as the Linux Bridge or switch software OpenFlow.

- **Hosts**: In Mininet, it is simply a shell process (e.g. bash) located within its own namespace network. Each host has its own virtual Ethernet interface (built and installed with `ip link add / set`) and a pipe to a Mininet father process, `mn`, which sends commands and monitors the output.

- **Switches**: OpenFlow switches provide the same semantic software package delivery that a physical device provides.

- **Controllers**: A controller can be in any part of the network (real or simulated), as long as the machine where the switches are running has connectivity to IP level with the controller. While Mininet running in a virtual machine, the controller may be in the own virtual machine, natively on the host computer or in the cloud.

Interaction

Once the network is initialized, it is possible to interact with it, for instance, executing commands on different hosts, switch checking operations, introducing bugs or adjusting connectivity. For this, Mininet includes a CLI, and is aware of the existence of the network in a manner that allows developers to control and manage, for example, a network from a console.

Customization

Mininet exports a Python API to create customized topologies and nodes (switch, controller, or other hosts). A few lines of code are enough to define a regression testing that, as the network is created, runs a command on multiple nodes, and shows results.

4.4.2 Study of a network topology

There are different network topologies predefined in the system. This section will focus on the study of a simple topology created in order to get started and understand the key functions. This is a topology that includes a switch connected to two hosts and an OpenFlow controller.
The first step to start a study of any network is its load on the system. To do this, once logged into the virtual machine, one must run this command:

$ sudo mn

With this command the different components will be loaded, links will be established and the CLI Mininet will appear.

**Figure 4.7: Scenario 1 switch and 2 hosts**

**Figure 4.8: sudo mn command**

**Getting information about the network**

First of all, a study of the most basic functions that Mininet offers. These functions provide information on different network nodes.
• $ nodes: these list the components that are part of the network.

• $ dump: this provides basic information about each node, e.g. the IP address and the PID code.

• $ net: this displays the network links.

\[\text{FIGURE 4.9: nodes, dump and net commands}\]

• ifconfig: this provides more detailed information about nodes. It is necessary to specify the element.

\[\text{FIGURE 4.10: h1 ifconfig command}\]
4.4.3 Python API Examples

Mininet offers a set of Python implementations that were designed as an aid for beginners [18]. This section will discuss some of the most important implementations:

- **consoles.py**: this example generates a set of windows, one for each node, allowing the interaction and monitoring of each of them. It also allows all sorts of tests on various network components.

- **emptynet.py**: This code generates an empty network and then adds other nodes to it.

- **linearbandwidth.py**: This example shows how to create a custom topology programmatically by using Topo subclasses and also how to run a series of tests on it.

- **miniedit.py**: shows how to create a network through a graphical editor.

- **multitest.py**: creates a network and runs several tests on it.

- **scratchnet.py and scratchnetuser.py**: These two examples show how to create a network using low-level functions of Mininet.

- **tree1024.py**: creates a 1024 hosts network and then the CLI runs with it.
• treeping64.py: this example creates a tree network with 64 hosts. The connectivity is tested using the ping function for three different types of switch / datapath.

4.4.4 Creating a virtual network topology

This point aims to show how to develop a Mininet topology using the Python API. To do this, a simple scenario has been created in order to properly understand the performing of a new network topology. It consists of two switches with three hosts each.

The code is the following:

```python
from mininet.topo import Topo

class MyTopo( Topo):
    "Simple topology example."

def __init__( self ):
    "Create custom topo."

    # Initialize topology
    Topo.__init__( self )

    # Add hosts and switches
    leftHost1 = self.addHost( 'h1' )
    leftHost2 = self.addHost( 'h2' )
    leftHost3 = self.addHost( 'h3' )
    rightHost4 = self.addHost( 'h4' )
```

Figure 4.12: Network topology with 2 switches and 6 hosts
rightHost5 = self.addHost( 'h5' )
rightHost6 = self.addHost( 'h6' )
leftSwitch = self.addSwitch( 's1' )
rightSwitch = self.addSwitch( 's2' )

# Add links
self.addLink( leftHost1, leftSwitch )
self.addLink( leftHost2, leftSwitch )
self.addLink( leftHost3, leftSwitch )
self.addLink( leftSwitch, rightSwitch )
self.addLink( rightSwitch, rightHost4 )
self.addLink( rightSwitch, rightHost5 )
self.addLink( rightSwitch, rightHost6 )

topos = { 'mytopo': ( lambda: MyTopo() ) }

Once the file is generated and stored within Mininet, the code must be executed to verify it. The
next command runs the topology topo-2sw-6host.py, which is located in mininet/custom.

$ sudo mn –custom /mininet/custom/topo-2sw-6host.py –topo mytopo

The functions explained in the previous section, among others, could be implemented to study
the topology.

![Terminal](image)

**FIGURE 4.13: Virtual network topology example**
Chapter 5

OpenDaylight

This chapter explains what OpenDayLight is. It is an opensource project which aims to accelerate and promote innovation in the creation and adoption of an open and transparent SDN. OpenDaylight was created as a collaborative project based in an open code, making SDN and NFV compatible with all networks of all scales and sizes. This proposal aims to emulate what Hadoop did within the big data market or OpenStack in cloud computing. This means that it becomes an open platform used by all enterprises. In that way, proprietary systems will not limit the market growth and the development costs will be reduced.

Particularly, this project assumes the role of an open platform which can be used by all enterprises. The system will not limit market growth and will in turn reduce the development costs. In the last few years virtualization and cloud technologies have been the innovative protagonists. Today, however, the networks have come to the fore. Until now, they have remained static and hardware-based. This change occurred with the creation of SDN, providing the increased ability to scale and program with more flexibility. This was due to the removal of the intelligence of the routers and switches, which was given to controllers based in software.

5.1 OpenDaylight, a first view

The OpenDaylight project is a collaborative open source project sponsored by the Linux Foundation. The objective of the project is to accelerate the adoption of SDN and create a solid base for Network functions Virtualization (NFV).
The first release was published in February 2014 under the name of Hydrogen. This release consists of three different editions: base, virtualization and Service provider. Those interested in exploring SDN and OpenFlow for proof-of-concepts or academic initiatives either in physical or virtual environments are meant to use Hydrogen Base Edition. Virtualization Edition includes the same features as Base Edition but adding functionality for creating and managing Virtual Tenant Networks and virtual overlays plus applications for security and network management. In the other hand, Service Provider Edition is for providers and carriers who manage existing networks and want to plot a path to SDN and NFV. The second and last release "Helium" has a new user interface and a simpler installation process, due to the Apache Karaf container.

5.1.1 Current Projects, an overview

As OpenDaylight is structured in different projects in order to handle the different software features provided, this section provides a quick overview of some of the ongoing projects with their aims. For more detailed information about the OpenDaylight project list visit the OpenDaylight wiki[19].

- **AAA** - will develop capabilities for Authentication, Authorization and Accounting.

- **BGP LS PCEP**: brings a south-bound plugin for supporting BGP Linkstate Distribution as a source of L3 topology information and another one to add support for Path Computation Element Protocol as a way to instantiate paths into the underlying network.

- **Documentation Group**: aims to support the OpenDaylight projects with improved documentation.

- **Integration Group** – coordinates and drives integration and test efforts to achieve a successful OpenDaylight release.

- **Defense4All** – is an application that detects and mitigates Distributed Denial of Service (DDoS) attacks.

- **Group Policy** – separates application connectivity requirements and underlaying details of network infrastructure.

- **Controller project**: is, without a doubt, the most important project. It is a highly available, modular, extensible, scalable, and multi-protocol Controller infrastructure built for SDN deployment on modern heterogeneous multi-vendor networks.

- **Dlux** - This project focuses on OpenDaylight’s user experience.
Chapter 5. OpenDaylight

- **List Mapping service** – provides a flexible map-and-encap framework that can be used for overlay network applications.

- **OpenDove** - is a network virtualization platform with a full control plane implementation for Open Daylight and data plane based on Open vSwitch.

- **OpenFlow Plugin** - intends to develop a plugin to support OpenFlow implementations and specifications.

- **OpenFlow Protocol Library** - provides a communication channel to the user in order to provide a mechanism for managing network hardware devices.

- **OVSDNB Integration Plugin** – accomplishes the implementation of the Open vSwitch Database management.

- **Reservation** – develops a SNMP (Simple Network Management Protocol) southbound plugin to control supporting SNMP devices.

- **SNMP4SDN** – provides end to end multi-layer provisioning. It focuses on flexible provisioning.

- **Virtual Tenant Network project (VTN)** - is an application that provides multi-tenant virtual network on an SDN controller.

- **Yang Tools** – is an infrastructure project aiming to develop necessary tooling and libraries providing support of NETCONF and YANG for Java (JVM-language based) projects and applications, such as Model Driven SAL for Controller (which uses YANG as it’s modeling language) and Netconf / OFConfig plugin.

### 5.2 Technical aspects

OpenDaylight is a community-led, open, industry-supported framework, for accelerating adoption, fostering new innovation, reducing risk and creating a more transparent approach to Software-Defined Networking[20].

SDN is usually is described in layers. The following figure displays a more detailed view of these layers, including the projects, services, applications and plugins that OpenDaylight provides:

As mentioned in Chapter 3, SDN is divided in three layers:

**Network Applications and Orchestration (NorthBound interface)**

The top layer is the application layer, which consists of business and network logic applications,
NorthBound APIs are located in this layer. The function of these NB APIs is controlling and managing the network and communicate it to the OpenDaylight controller, which makes the appropriate decisions and informs the lower layer using the SouthBound APIs.

**Controller Platform**

The middle layer is composed by the OpenDaylight controller. It is in this layer where SDN abstraction can manifest and implement the OpenFlow protocol to communicate and control the switching elements of the network (bottom layer) using the SouthBound APIs. With this APIs the controller is able to make changes according to real-time demands and needs.

**Network Devices (southbound interface)**

The bottom layer consists of the physical & virtual network infrastructure (devices, switches, routers and transmission media), that create the connective fabric between all endpoints within the network.
5.3 Controller Project[21]

OpenDaylight contains - among other projects - a controller platform, modular, extensible, scalable, multi-protocol and flexible. This controller is strictly implemented in software and enclosed in its own java virtual machine (JVM). Thus, it can be used in any device or platform supporting Java.

The platform controller contains a number of dynamic modules to execute the necessary networks tasks. There are a number of basic network services for these tasks, such as understanding what devices are contained in the network and the capabilities of each one of them, or statistic compilation, etc. In addition, platform-oriented services and other extensions can be incorporated into the controller platform to improve SDN functionality.

![Controller architecture](image)

**Figure 5.2: Controller architecture**

The southbound interface is capable of supporting multiple protocols (as separate plugins), eg OpenFlow 1.0, 1.3 OpenFlow, BGP-LS, etc. These modules are dynamically linked with the Service Abstraction Layer (SAL).

The figure below illustrates the controller architecture.
5.3.1 Service Abstraction Layer

SAL provides the necessary abstraction to support multiple southbound protocols through plugins. The northbound extensible architecture application oriented provides a rich set of Northbound APIs through RESTful web services and OSGi services.

It is at the heart of the controller modular design which supports multiple protocols to southbound and provides consistent services for modules and applications (where the business intelligence resides). SAL determines as the requested services must be fulfilled regardless of the underlying protocol used between the controller and network devices. Basically, adapts the different technologies used in the southbound. In order to make the controller able to administer the devices, it needs to know about these devices, their capabilities, accessibility, etc. This information is stored and managed by the Topology Manager. Other components such as ARP Handler, Host Tracker and the Switch Manager Device help to generate the database topology for the Topology Manager.

![Service Abstraction Layer Diagram](http://www.opendaylight.org)

It provides basic services as device discovery, which are used by modules such as the Topology Manager to build the topology and device capabilities. Services are built using the features exposed by plugins (based on the presence of a plugin and capabilities of network device). Based on the service request, SAL maps it to the appropriate plugin and therefore to the most
appropriate Southbound protocol to interact with the network device given. As a reminder, each plugin is independent to another.

<table>
<thead>
<tr>
<th>Bundle</th>
<th>Exported interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arphandle</td>
<td>IHostFinder</td>
<td>Component responsible for learning about host location by handling ARP</td>
</tr>
<tr>
<td>hosttracker</td>
<td>IfIptoHost</td>
<td>Track the location of the host relatively to the SDN network</td>
</tr>
<tr>
<td>switchmanager</td>
<td>ISwitchManager</td>
<td>Component holding the inventory information for all the known nodes (i.e., switches) in the controller.</td>
</tr>
<tr>
<td>topologymanager</td>
<td>ITopologyManager</td>
<td>Component holding the whole network graph</td>
</tr>
<tr>
<td>usermanager</td>
<td>IUserManager</td>
<td>Component taking care of user management.</td>
</tr>
<tr>
<td>statisticsmanager</td>
<td>IStatisticsManager</td>
<td>Component in charge of using the SAL ReadService to collect several statistics from the SDN network.</td>
</tr>
<tr>
<td>sal</td>
<td>IReadService</td>
<td>Interface for retrieving the network node’s flow/port/queue hardware view</td>
</tr>
<tr>
<td>sal</td>
<td>ITopologyService</td>
<td>Topology methods provided by SAL toward the applications</td>
</tr>
<tr>
<td>sal</td>
<td>IFlowProgrammerService</td>
<td>Interface for installing/modifying/removing flows on a network node</td>
</tr>
<tr>
<td>sal</td>
<td>IDataPacketService</td>
<td>Data Packet Services SAL provides to the applications</td>
</tr>
<tr>
<td>web</td>
<td>IDaylightWeb</td>
<td>Component tracking the several pieces of the UI depending on bundles installed on the system.</td>
</tr>
</tbody>
</table>

**TABLE 5.1: Important bundles OpenDaylight**

### 5.3.1.1 From AD-SAL to MD-SAL

At first, static APIs were used. With API-Driven SAL (AD-SAL), request routing and service adaptation were performing. That is, when a NorthBound plugin wishes to talk to the SouthBound plugins, is this SouthBound plugin the one doing the request. Regarding the adaptation between NB and SB services, an existing code runs in the SB interface in order to provide this adaptation. However, with the emergence of Model Driven SAL (MD-SAL), the APIs are automatically generated using YANG models and, always maintaining the compatibility with previous versions. In MD-SAL the service adaptation is yield in an adaptation plugin located
outside the SAL. Firstly, the NB service model needs to be defined for then, the adaptation
plugin is loaded into the controller. Consequently, the SAL is completely independent of the
adaptation process. The request routing is preserved.

The following table illustrates the main differences between AD-SAL and MD-SAL.

**Figure 5.4: From AD-SAL to MD-SAL**  
Source: https://wiki.opendaylight.org  MD-SAL App Tutorial
The SAL APIs, request routing between consumers and providers, and data adaptations are all statically defined at compile/build time.

The AD-SAL typically has both NB and SB APIs even for functions/services that are mapped 1:1 between SB Plugins and NB Plugins.

In AD-SAL there is a "dedicate" REST API for each northbound/southbound plugin.

The AD-SAL provides request routing (selects an SB plugin based on service type) and optionally provides service adaptation, if an NB (Service, abstract) API is different from its corresponding SB (protocol) API.

Request routing is based on plugin type: the SAL knows which node instance is served by which plugin, and when an NB Plugin requests an operation on a given node, the request is routed to the appropriate plugin which then routes the request to the appropriate node.

AD-SAL is stateless

Limited to flow-capable device and service models only

AD-SAL services usually provide both asynchronous and synchronous versions of the same API method

<table>
<thead>
<tr>
<th>AD-SAL</th>
<th>MD-SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>The SAL APIs, request routing between consumers and providers, and data adaptations are all statically defined at compile/build time.</td>
<td>The SAL APIs, request routing between consumers and providers are defined from models, and data adaptations are provided by ‘internal’ adaptation plugins.</td>
</tr>
<tr>
<td>The AD-SAL typically has both NB and SB APIs even for functions/services that are mapped 1:1 between SB Plugins and NB Plugins</td>
<td>The MD-SAL allows both the NB plugins and SB plugins to use the same API generated from a model. One plugin becomes an API (service) provider; the other becomes an API (service) Consumer.</td>
</tr>
<tr>
<td>In AD-SAL there is a &quot;dedicate&quot; REST API for each northbound/southbound plugin</td>
<td>MD-SAL provides a &quot;common&quot; REST API to access data and functions defined in models.</td>
</tr>
<tr>
<td>The AD-SAL provides request routing (selects an SB plugin based on service type) and optionally provides service adaptation, if an NB (Service, abstract) API is different from its corresponding SB (protocol) API</td>
<td>The MD-SAL provides request routing and the infrastructure to support service adaptation, but it does not provide service adaptation itself; service adaptation is provided by plugins.</td>
</tr>
<tr>
<td>Request routing is based on plugin type: the SAL knows which node instance is served by which plugin, and when an NB Plugin requests an operation on a given node, the request is routed to the appropriate plugin which then routes the request to the appropriate node</td>
<td>Request Routing in the MD-SAL is done on both protocol type and node instances, since node instance data is exported from the plugin into the SAL.</td>
</tr>
<tr>
<td>AD-SAL is stateless</td>
<td>MD-SAL can store data for models defined by plugins: provider and consumer plugins can exchange data through the MD-SAL storage.</td>
</tr>
<tr>
<td>Limited to flow-capable device and service models only</td>
<td>Model agnostic. It can support any device and/or service models and is not limited to flow-capable device and service models only.</td>
</tr>
<tr>
<td>AD-SAL services usually provide both asynchronous and synchronous versions of the same API method</td>
<td>In MD-SAL, Service model APIs only provide asynchronous APIs, but they return a ‘java.concurrent.Future’ object, which allows a caller to block until the call is processed and a result object is available. Same API can be used for both synchronous and asynchronous approach. Thus MD-SAL encourages asynchronous approach to application design but does not preclude synchronous applications.</td>
</tr>
</tbody>
</table>

TABLE 5.2: Differences between AD-SAL and MD-SAL
Source: http://sdntutorials.com/difference-between-ad-sal-and-md-sal
Chapter 6

OpenDaylight Application Development

6.1 Background

After explaining what OpenDaylight is, now is the moment to describe how to write an OpenDaylight application. Basically, an OpenDaylight application can be developed using two broad procedures: using the OSGi interfaces to (Java APIs), that is writing a "native" application; otherwise an application can be written using the RESTful APIs, which is not necessarily native to the framework.

With OSGi interfaces the application can exploit much more functionality as well as have higher performance. However, using the RESTful way, an application can be agnostic to the complexities associated with the OpenDaylight framework as well as being completely language independent (See Figure 6.1). Deciding which way should be use to write an application depends on the developer and his/her priorities: flexibility and performance (OSGi interfaces) or ease and turnaround time (RESTful APIs).

Before starting with how to develop an OpenDaylight application a few concepts are required to be defined.

**OSGi**

The OSGi (Open Service Gateway Interface) is a layer of Java that allows to create modules that can interact runtime and it forms the backend of the OpenDaylight platform. With an OSGi framework an environment for the modularization of applications is provided into smaller bundles, OSGi describes a modular system and service platform in Java that implements an entire and changing component model.
Maven and maven-bundle plugin
Maven is used for OpenDaylight to build automation and dependency management. Maven is a software tool that allows managing and building Java-based projects, based in XML format. Maven-bundle-plugin provides a maven plugin that supports creating an OSGi bundle.

POM file
Project Object Model (POM) is an XML file that contains information about the project and configuration details used by Maven to build the project. It is named as pom.xml, and is typically located in the root directory of a project. However, more than one pom.xml files can appear if the project is divided into subprojects, where there will be one for the parent project and one in every subproject, in order to build the whole project or only the subproject, as the developer desires.

Karaf
Apache Karaf is an OSGi based runtime, which provides a slight container, within which different elements and applications can be deployed. It is a very malleable and extensible container.

YANG
In order to generate compatible interfaces for a model driven plugin a modelling language and YANG Tool are used. These models are defined in YANG (Yet Another Next Generation), that once compiled generates definitions for distinct APIs applicable for the plugin. As Wikipedia says “YANG is a data modeling language for the NETCONF network configuration protocol”. The logic can be written by developer thanks to provided templates.
RESTful API

REST (Representational State Transfer) is a software architecture style for managing state information. REST entirely relies on the HTTP standard, which allows creating services and applications that can be used by any device or client who understand HTTP. This proposal is simpler and more conventional than other alternatives, such as SOAP or XML-RPC.

6.2 Writing an Application using OSGi interfaces

There are two existing methodologies to write an application using the OSGi Interfaces: Application Driven (AD-SAL) or Model Driven (MD-SAL). As explained in section 5.3.1, the first one focuses on development of a standalone service consumer, consuming services produced by basic or domain-specific service plugins. Whereas, Model Driven methodology does it through the use of YANG based models. Coming up next these two processes will be describe in detail.

6.2.1 AD-SAL Application Development

In order to study and understand properly the high-level steps for developing of an AD-SAL compliant plugin in OpenDaylight, the SDN Hub tutorial[22] has been followed, and an example of application has been created. Before starting with the technical aspects of the application development, the following scenario is proposed:

6.2.1.1 Scenario

Our network consists in a single switch (s1) with three hosts connected to it (h1, h2 and h3). Unfortunately, there is a malicious host (h3) that wants to saturate a website launching a DoS attack to consume SDN resources of the network, so that the other two devices (h1 and h2) will fail to load the website.

A Denial of Service (DoS) attack aims to exhaust resources on vulnerable switches and/or the SDN controller, hereby affecting forwarding in the data plane for an indefinite period of time. It consists basically of sending a large number of requests to a server so that users of the service are not able to access these resources.

A framework has been used to detect the attack on the network topology and data plane forwarding. This example proposes a solution establishing the traffic flow uniquely between h1 and h2, blocking the communication with h3.
Mininet is the network emulator used to create this virtual network topology. This topology is already implemented in mininet as ” —topo single,3”.

To write this example, the next steps have been followed:

6.2.1.2 IDE Setup

First of all, while not mandatory, setting up an integrated development environment (IDE) for OpenDaylight, in this case a Maven-Eclipse integration[23], which basically includes the Java compiler and the Maven build tools. An IDE (Integrated Development Environment) is recommended as it eases the process of viewing and editing the source code and command-line for compiling and running the controller.

6.2.1.3 POM XML File

The next step consists of creating the pom.xml file explained above, which indicates the inter-bundle dependencies and its specifications. With the pom.xml file, bundle specifications would define the required services by the plugin.

Part of the pom.xml file[24] is shown below. The structure of this file can be divided in four sections:
• Parent definition

• Import packages (required for AD-SAL) and Export packages (AD-SAL application for other packages) definition

• Bundle activator definition

• Dependencies definition

<project>
  ...
  <parent>
    <groupId>org.sdnhub.odl.tutorial</groupId>
    <artifactId>commons</artifactId>
    <version>1.1.0-SNAPSHOT</version>
    <relativePath>../commons/parent/</relativePath>
  </parent>

  <artifactId>adsal_L2_forwarding</artifactId>
  <version>0.5.0-SNAPSHOT</version>
  <name>L2 forwarding tutorial with AD-SAL OpenFlow plugins</name>
  <packaging>bundle</packaging>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.3.6</version>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Import-Package>
              org.opendaylight.controller.sal.core,
              org.opendaylight.controller.sal.utils,
              ...
              org.osgi.framework
            </Import-Package>
            <Export-Package>
              org.opendaylight.tutorial.tutorial_L2_forwarding
            </Export-Package>
            <Bundle-Activator>
              org.opendaylight.tutorial.tutorial_L2_forwarding.internal.Activator
            </Bundle-Activator>
          </instructions>
          <manifestLocation>${project.basedir}/META-INF</manifestLocation>
        </configuration>
      </plugin>
    </plugins>
  </build>
6.2.4 Activator Java

Along with pom.xml file, activator.java is another important file in the setting up environment process. It could be described as a template that handles the configuration of the dependencies during run-time. Here a piece of the Activator.java file[25]:

```java
public class Activator extends ComponentActivatorAbstractBase {
    protected static final Logger logger = LoggerFactory.getLogger(Activator.class);

    public void init() { ... }
    public void destroy() { ... }
    public Object[] getImplementations() {
        Object[] res = { TutorialL2Forwarding.class};
        return res;
    }

    public void configureInstance(Component c, Object imp, String containerName) {
        if (imp.equals(TutorialL2Forwarding.class)) {
            // export the services
            Dictionary<String, String> props = new Hashtable<String, String>();
            props.put("salListenerName", "tutorial_L2_forwarding");
            c.setInterface(new String[] { IListenDataPacket.class.getName() }, props);

            // register dependent modules
            c.add(createContainerServiceDependency(containerName).setService(
                ISwitchManager.class).setCallbacks("setSwitchManager",
                "unsetSwitchManager").setRequired(true));
            ...
        }
    }
}
```

6.2.5 Plugin Code

Now the environment is ready for the plugin code implementation. Here is where the logic of our code resides and where the services of OpenDaylight bundles are imported. The next example is based on the example single switch solution of the learning switch provided by SDN Hub.

This application receives packets sent by the switch to the controller by implementing the ListenDataPacket service of SAL.
There are a few important methods that need to be highlighted to explain the process of the development of this application example.

First of all, the method receiveDataPacket is invocated to check the MAC source and destination addresses.

Then, the IListenDataPacket interface calls this method when when a packet arrives at a node without a pre-cached rule and the dataPacketService is used to forward a packet. This service sends a packet out of a node-connector.

After forwarding and processing the packet, the other bundles are informed through PacketResult.CONSUME. The packets sent to the host 3 (ip=10.0.0.3) are ignored and the receiveDataPacket method returns PacketResult.IGNORED.

The functions getSourceMACAddress() and getDestinationMACAddress() are used to parse packet information.

Finally, the receiveDataPacket method needs to create an action list and consequently, match them.

Here we can see the logic of our application:

```java
public class TutorialL2Forwarding implements IListenDataPacket{

    @Override
    public PacketResult receiveDataPacket(RawPacket inPkt) {
        if (inPkt == null) {
            return PacketResult.IGNORED;
        }

        // Use DataPacketService to decode the packet.
        Packet pkt = this.dataPacketService.decodeDataPacket(inPkt);

        // Extract the incoming node (i.e., switch) and node-connector (i.e., switchport)
        NodeConnector incoming_connector = inPkt.getInIncomingNodeConnector();
        Node incoming_node = incoming_connector.getNode();

        if (pkt instanceof Ethernet) {
            Object next = pkt.getPayload();

            if (next instanceof IPv4) {
                for (int i = 0; i <= 1; i++) {
                    IPv4 ipv4Pkt = (IPv4) next;
                    InetAddress dstIP = intToInetAddress(ipv4Pkt.getDestinationAddress());
                    if (dstIP.equals(virtual_ip3)) {
```
return PacketResult.IGNORED;
}
}
}
else return PacketResult.IGNORED;

byte[] srcMAC = ((Ethernet) pkt).getSourceMACAddress();
long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
this.mac_to_port.put(srcMAC_val, incoming_connector);

byte[] dstMAC = ((Ethernet) pkt).getDestinationMACAddress();
long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
NodeConnector outgoing_connector =
this.mac_to_port.get(dstMAC_val);

if (!outgoing_connector.equals(incoming_connector)) {
    try {
        RawPacket destPkt = new RawPacket(inPkt);
        destPkt.setOutgoingNodeConnector(outgoing_connector);
        this.dataPacketService.transmitDataPacket(destPkt);
    } catch (ConstructionException e2) {
        logger.warn("Could not create packet for PACKET_OUT");
        return PacketResult.CONSUME;
    }
}

Match match = new Match();
match.setField(MatchType.IN_PORT, incoming_connector);
match.setField(MatchType.DL_DST, dstMAC.clone());
match.setField(MatchType.DL_TYPE, EtherTypes.IPv4.shortValue());
match.setField(MatchType.IN_PORT, incoming_connector);

NodeConnector dst_connector;

// Do I know the destination MAC?
if (((dst_connector = this.mac_to_port.get(dstMAC_val)) != null) {
    List<Action> actions = new ArrayList<Action>();
    actions.add(new Output(dst_connector));
    Flow f = new Flow(match, actions);
    f.setIdleTimeout((short) 10);

    // Modify the flow on the network node
    Status status = programmer.addFlow(incoming_node, f);
    if (!status.isSuccess()) {
        logger.warn("SDN Plugin failed to program the flow: {}. The failure is: {}", f, status.getDescription());
        return PacketResult.IGNORED;
    }
    logger.info("Installed flow {} in node {}", f, incoming_node.getIdentifier());
}
6.2.1.6 Evaluation test

Once the code has been developed, OpenDaylight OSGi framework compiles the build using
the Maven tool, which at the same time uses the POM file to do it.

After the code was written, the subsequent steps were followed to run the application:

- Run the following command to create the virtual network topology using Mininet and
  Open vSwitch. This command emulates the desired topology (1 single switch with 3
  hosts), that attempts to connect to a remote controller (OpenDaylight):

  $ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller remote

- If we run the command "pingall", we can see that there is no reachability between
  hosts as the switch does not have the intelligence to learn the MAC addresses of each host
  and to forward traffic to the correct switch ports.

  ![Pingall unreachable](image)

  **Figure 6.3: pingall unreachable**

- To add intelligence to the switch, the application needs to be built and inserted into the
  OSGi platform. As previously mentioned, it is achieved using maven tool. In a new ter-
  minal, in the folder where the pom.xml file resides, run the following command:

  $ mvn install -nsu
- Run the controller itself to start the OSGi bundles installed as jar files in the plugins directory. With the controller running and after all bundles are started, if we go back to the Mininet terminal we can see that our application works as expected, hosts h1 and h2 "talk" to each other, but h3 remains outside the "conversation".

![Image of Mininet terminal results](image1.png)

**Figure 6.4: Results**

### 6.2.2 MD-SAL Application Development

Thanks to YANG models, APIs are automatically generated and therefore MD-SAL which entails an easier application development for the controller. Therefore, MD-SAL has been well accepted by developers’ community.

However, it requires a deeper understanding. This section will attempt to briefly explain what MD-SAL is and how to develop an MD-SAL application.

![Image of MD-SAL software architecture](image2.png)

**Figure 6.5: MD-SAL Software Architecture View**

Source: [https://wiki.opendaylight.org](https://wiki.opendaylight.org) MD-SAL App Tutorial
Until MD-SAL appeared, the general overview presented consisted of protocol plugins in the bottom, services plugins at the top and adaptations to these plugins in the middle. In MD-SAL, the Service Abstraction Layer acts as a backbone of the system, where the plugins are plugged into. That means, that the SAL is being just a plum connecting "consumers" and "providers", it is an independent infrastructure.

**Pre-requisites**: to avoid future problems, it is necessary to have installed the Java 1.7.0 version instead of the newer Java 1.8.0. Also, and Apache Maven version higher than 3.1.1 version is required and Eclipse as IDE (see section 6.2.1.2).

### 6.2.2.1 POM XML file

As explained in section 6.2.1.3, the pom.xml file describes the dependencies to write against the flow capable model, to compile the sample against different models (e.g. inventory, flow services, flow base models) and also the MD-SAL APIs.

### 6.2.2.2 Activator Java

The bundle setup is already explained in section 6.2.1.3. For a better understanding the example provided by SDN Hub MD-SAL "learning switch" is used

To start the application, the helper class AbstractBindingAwareConsumer is extended:

```java
public class Activator extends AbstractBindingAwareConsumer implements AutoCloseable {

    // Implement onSessionInitialized method

    public void onSessionInitialized(ConsumerContext session) {
        // Call different dependencies
        DataBroker dataBroker = session.getSALService(DataBrokerService.class);
        RpcService rpcService = session.getSALService(RpcService.class);
        NotificationService notificationService = session.getSALService(NotificationService.class);
    }
}
```

The different dependencies need to be called: DataBroker, which provides access to other data tree and is used to configure the flows and to look up into the operational interface. RpcService is the packet processing service used to send the packets out. And the NotificationService, used to subscribe for the notifications. Basically, when the PacketProcessingService is get, the learningSwitchManagerMultiImpl class is started.

```java
learningSwitch = new LearningSwitchManagerMultiImpl();
learningSwitch.setDataBroker(session.getSALService(DataBrokerService.class));
```
6.2.2.3 Plugin Code Overview

MD-SAL uses the yang models. These models define the data structures, RPCs and notifications. Basically, plugins (or applications) are built for the controller while defining a new model. There are a set of tools (e.g. Maven Plugin) that take the YANG model and generate Java interfaces in the compile time. These Java interfaces represent the RPCs of the data structures. Summarizing, the developer has an API that is compiled into the Jar file to describe the model. In the run-time, these jar files also contain the model and it is used for other MD-SAL functions. The plugin must is developed against these generated APIs and against a set of MD-SAL APIs (where services are registered).

Finally, maven is used to compile the plugin. It provides a jar file that needs to be deployed to the controller to run the application.

6.2.2.4 Prepare the project

To get started on creating a MD-SAL application, maven project templates are used. These templates are known as maven archetype or MD-SAL-App-Simple.
Run the Maven Archetype
The necessary steps to run the Maven Archetype are explained in the OpenDaylight wiki [26].

After executing the commands, some parameters will be asked: groupid, artifactid, version, package and modelFields. At this point, the generation of the maven archetype is completed.

For more information about Apache Maven Archetypes visit this site [27]

Finish Generation – Run the 'Generator'
As maven archetypes are extremely limited in their capabilities, one more step is necessary. A remainder of the template code is generated for the sample application. To do this we need to make use of the "generate" folder which was created in the previous step.

Now, the project can be imported to the IDE.

What the MD-SAL Archetype Provides
As can be appreciated at the end of the execution of the "Run the Maven Archetype", 5 subfolders were created under the <artifactId> folder.

- **Consumer**: how an application can be written, consuming the RPC (Remote Procedure Calls) services provided with the provider bundle.

- **Generate**: helper folder used to add more capabilities to the maven archetypes.

- **Model**: sample yang file defining the "model".

- **Provider**: it sets up a service that implements any RPC already defined in the model’s yang files.

- **Web**: application that allows defining customer REST APIs.

- **Features**: list of sample features to enable provider, consumer and web bundles.

Additional Generation Parameters
Naming the application and provide the Artifac and Group IDs are optionals, the command line for these parameters are the followings:
-DappName=<appNamePrefix>
-DgroupId=«Any group Id such as org.opendaylight.controller»
-DartifactId=«Any app name such as mdSalExample»

**How to Remove Unneeded Projects**

Follow the next steps:

- Delete the project: rm –rf ’project’
- Modify the pom.xml in the root folder
- Remove the unneeded project from the ”modules” section
- Remove the feature of the unneeded project, it is located in features/src/main/resources directory
- Run ’mvn clean install’

**Advanced Usage of the MD-SAL-App-Simple Archetype**

For a better auto generation of a yang file there are advanced features of the MD-SAL-App-Simple archetype. To do this you will make use of the ”modelProperties” field.

When prompted during the interactive mode, or optionally using the ’-DmodelFields’ flag on the command line, you can provide a JSON formatted list of fields that you want included in your yang file.

**6.2.2.5 POM XML File**

As explained in section 6.1, the pom.xml file describes the dependencies to write against the flow capable model, to compile the sample against different models (e.g. inventory, flow services, flow base models) and also the MD-SAL APIs.

**6.2.2.6 Karaf Feature**

Apache Karaf is a small OSGi based runtime which provides a lightweight container for loading different modules. In Apache Karaf, the application provisioning is an Apache Karaf ”feature”.

Using karaf feature an application is described as a name, a version and a set of bundles. Optionally, an application can be also described providing a description, a set of configurations or configuration files and a set of dependency features.

When you install a feature, Apache Karaf installs all resources described in the feature. It means that it will automatically resolves and installs all bundles, configurations, and dependency features described in the feature. For further information about Karaf Features visit the following site[28].

Define a Karaf Feature

Karaf features are defined in the src/main/resource/features.xml, located in the previously created 'feature' folder. For a better understanding, the md-sal application sample of SDN Hub tutorial is used. In the following code snipped, we can see how the "thesis" feature is added.

```xml
<feature name='odl-tesis' version='${project.version}'>
  <feature version='${yangtools.version}'>odl-yangtools-common</feature>
  <feature version='${yangtools.version}'>odl-yangtools-binding</feature>
  <feature version='${project.version}'>odl-mdsal-broker</feature>
  <bundle>mvn:org.opendaylight.controller.samples/thesis/${project.version}</bundle>
  <bundle>mvn:org.opendaylight.controller.samples/thesis-consumer/${project.version}</bundle>
  <bundle>mvn:org.opendaylight.controller.samples/thesis-provider/${project.version}</bundle>
  <configfile finalname='${config.configfile.directory}/${config.thesis.configfile}'>
    mvn:org.opendaylight.controller.samples/thesis-config/${project.version}/xml/config</configfile>
</feature>
```

The following step for defining a Karaf Feature is, again, using the pom.xml contained in the features project. A maven-resources-plugin is created to make the features.xml available outside the jar in the maven repository.

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-resources-plugin</artifactId>
  <executions>
    <execution>
      <id>filter</id>
      <goals>
        <goal>resources</goal>
      </goals>
      <phase>generate-resources</phase>
    </execution>
  </executions>
</plugin>
```
Once the features have been defined in the pom.xml, the maven Project can be built just running the command:

$ mvn clean install

**Feature Dependency**
While developing a feature, it may use (depend on) services such as RPCs or models. These features must be also installed to maven repository and specified in the dependent’s definition.

```xml
<repository>mvn:org.opendaylight.yangtools/features-yangtools/ ${ yangtools.version}/xml/features</repository>
<repository>mvn:org.opendaylight.controller/features-mdsal/ ${controller.mdsal.version}/xml/features</repository>
```

**Deploy the Features**
With the karaf distribution already running, execute the following command:

$ feature:repo-add mvn:<groupId>/features-<artifactId>/<version>/xml/features

Here [29] the installing instructions, how to run and how to add a feature.

After adding the feature, the next step consist in its installation:

$ feature:install 'feature_name'

To list the available features type:

$ feature:list

### 6.2.2.7 MD-SAL Model

**YANG Modelling Language**
YANG (see section 6.1) is a data modeling language used by the Network Configuration Protocol (NETCONF).

In order to prepare the bundle, a first yang file defining data model is created under the name thesis.yang. In this file the thesis attributes, container, RPC and notifications are defined. It is located under `src/main/yang` folder and its general structure would be as follows:

```yang
module thesis {
```
Chapter 6. OpenDaylight Application Development

```yang
yang-version 1;
namespace "opendaylight:sample";
prefix thesis;
revision "2010-11-20" { description "Thesis module in progress.";
}
```

**YANG Maven Plugin**

To compile the thesis.yang file, the `yang-maven-plugin`[30] needs to be specified in the pom.xml.

```xml
<plugin>
  <groupId>org.opendaylight.yangtools</groupId>
  <artifactId>yang-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>generate-sources</goal>
      </goals>
      <configuration>
        <yangFilesRootDir>src/main/yang</yangFilesRootDir>
        <codeGenerators>
          <generator>
            <codeGeneratorClass>org.opendaylight.yangtools.maven.sal.api.gen.plugin.CodeGeneratorImpl</codeGeneratorClass>
            <outputBaseDir>${salGeneratorPath}</outputBaseDir>
          </generator>
        </codeGenerators>
        <inspectDependencies>true</inspectDependencies>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.opendaylight.yangtools</groupId>
      <artifactId>maven-sal-api-gen-plugin</artifactId>
      <version>${yangtools.version}</version>
      <type>jar</type>
    </dependency>
  </dependencies>
</plugin>
```

**Generate the code**

After running the command `mvn clean install` to build the project, the following classes will be generated:

- **Thesis**: an interface that represents the thesis container with methods to obtain the leaf node data.
• **ThesisData**: The thesis container stores the data in the module.

• **$YangModuleInfoImpl**: information about the YANG module.

• **$YangModelBindingProvider**: private constructor of $YangModuleInfoImpl.

### 6.2.2.8 Implementing the Provider Plugin

This section aims to explain how to write the provider ‘thesis-provider-impl.yang’ to perform RPC implementation and to add notification listeners. However, MD-SAL dependencies must be included.

#### Service Dependency Configuration

Again, the yang-maven-plugin configured in the pom.xml is used to generate the Java code to help with wiring into the config subsystem:

```xml
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
      <instructions>
          <Export-Package>org.opendaylight.controller.config.yang.thesis_provider</Export-Package>
          <Import-Package>*</Import-Package>
      </instructions>
  </configuration>
</plugin>

<plugin>
  <groupId>org.opendaylight.yangtools</groupId>
  <artifactId>yang-maven-plugin</artifactId>
  <executions>
      <execution>
          <id>config</id>
          <goals>
              <goal>generate-sources</goal>
          </goals>
          <configuration>
              <codeGenerators>
                  <generator>
                      <codeGeneratorClass>
                          org.opendaylight.controller.config.yangjmxgenerator.plugin.JMXGenerator
                      </codeGeneratorClass>
                      <outputBaseDir>${jmxGeneratorPath}</outputBaseDir>
                      <additionalConfiguration>
                          <namespaceToPackage1>
                              urn:opendaylight:params:xml:ns:yang:controller==
                              org.opendaylight.controller.config.yang
                          </namespaceToPackage1>
                      </additionalConfiguration>
                  </generator>
              </codeGenerators>
          </configuration>
      </execution>
  </executions>
</plugin>
```
Config Subsystem Dependency Configuration

The thesis-provider-imple.yang file is used by the code generator to create Java classes. This listing shows the declaration of dependency on the three Config Subsystem Services:

```xml
<generator>
  <codeGeneratorClass>
    org.opendaylight.yangtools.maven.sal.api.gen.plugin.CodeGeneratorImpl
  </codeGeneratorClass>
  <outputBaseDir>${salGeneratorPath}</outputBaseDir>
</generator>
<inspectDependencies>true</inspectDependencies>
</configuration>

// This is the definition of the service implementation as a module identity
identity thesis-provider-impl {
  base config:module-type;

  // Specifies the prefix for generated java classes.
  config:java-name-prefix ThesisProvider;
}

// Augments the 'configuration' choice node under modules/module.
augment "#/config:modules/config:module/config:configuration" {
  case thesis-provider-impl {
    when "#/config:modules/config:module/config:type = thesis-provider-impl";

    //Wires dependent services into this class - RPC registry service
    container rpc-registry {
      uses config:service-ref {
        refine type {
          mandatory true;
          config:required-identity mdsal:binding-rpc-registry;
        }
      }
    }

    container notification-service {
      uses config:service-ref {
        refine type {
          mandatory true;
          config:required-identity mdsal:binding-notification-service;
        }
      }
    }

    //Wires in the data-broker service
    container data-broker {
```
Chapter 6. OpenDaylight Application Development

Compiling the project, two files will be generated under the directory `src/main/java`:

- **ThesisProviderModule** - concrete class whose `createInstance()` method provides the OpenDaylightThesis instance.

- **ThesisProviderModuleFactory** - concrete class, instantiated internally by MD-SAL which creates ThesisProviderModule instances.

### 6.2.2.9 How Karaf knows about the Config Subsystem Dependency

This process is similar to the 'features.xml' description section 6.2.2.6. The Config Subsystem Services are also contained in xml-files. Karaf use them to initialize a module in the config subsystem and make it available for use in a controller environment.

In `configuration/initia/03-thesis-sample.xml` file:

```
<snapshot>
  <required-capabilities>
    ...
  </required-capabilities>
  <configuration>
    <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <modules xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
        <module>
          <name>thesis-provider-impl</name>
        </module>
      </modules>
    </data>
  </configuration>
</snapshot>
```

### Maven Plugin

Now that a maven plugin is added in the `pom.xml` provider, this file can be included in the `features.xml` file.
6.2.2.10 Implementing Consumer Plugin

The consumer plugin only need to call the RPC implemented by the provider. The developing process is the same as the explained for the Provider Plugin. This is declared in its thesis-consumer-impl.yang. The YANG file is the following:

```yang
//augments the configuration,
augment "/config:modules/config:module/config:configuration" {
  
  case thesis-provider-impl {
    
    when "/config:modules/config:module/config:type = thesis-provider-impl";
    
    //Wires dependent services into this class - RPC registry service
    container rpc-registry {
      
      uses config:service-ref {  
        
        refine type {
          
          mandatory true;
          
          config:required-identity mdsal:binding-rpc-registry;
        }  
      }  
    }  
  }  
}
```
Generate the Code

The last step to do is adding the yang-maven-plugin in the pom.xml file.

When the project is compiled ($mvn clean install), the module code is generated under the name ThesisConsumerModule.java and it provides the entry point for our module.

```java
public AutoCloseable createInstance(){
    ThesisService service = getRpcRegistryDependency().getRpcService(thesisService.class);
    final ThesisConsumerImpl consumerImpl = new ThesisConsumerImpl(service);
}
```

This method is called when the plugin is loaded to karaf.

OpenDaylight Controller project provides a toaster sample explaining step-by-step how to develop a MD-SAL application [31]

### 6.3 Writing an Application using RESTful APIs

The Service Abstraction Layer provides an environment to pass messages between NorthBound to Southbound interface. As described in section 6.2, there are two different SAL concepts, AD-SAL and MD-SAL, used to develop an application using an OSGi interface. When writing an application using REST APIs, it may be said that there are also two types. Although the way AD-SAL and MD-SAL provide REST API is really similar, their usage and their URL address are different.

With AD-SAL the REST APIs is hardcoded in the source code and the requests are listened from port 8080, whereas in MD-SAL the automatic generation of the REST API makes the API generation process much more flexible and it uses the port 8181. However, the additional model generation increases the complexity.

In the following sections, the topology single,3 (1 switch – 3 hosts) will be used. Again, Mininet is the tool used to create the topology:

```bash
$ sudo mn --topo single,3 --mac --switch ovsk.protocols=OpenFlow13 --controller remote
```

Also, the controller must be running. To do this, download and install Helium-SR3 release[32] and execute the following commands:

```bash
$ cd distribution-karaf-0.2.3-Helium-SR3 $ /bin/karaf
```
With the controller running, there are few required karaf library:


6.3.1 AD-SAL REST API

Now that the controller is running and our topology is created we can get Data through REST Requests using cURL.

Note: a packet has been sent from h1 to h2 ($ h1 ping h2 –c1)

- GET Network Topology (See Figure 6.7): http://localhost:8080/controller/nb/v2/topology/default (user:admin password:admin)
- GET Flow statistics (See Figure 6.8): http://localhost:8080/controller/nb/v2/statistics/default/flow
- GET Switches (See Figure 6.9): http://localhost:8080/controller/nb/v2/switchmanager/default/nodes

![Figure 6.7: GET Network Topology AD-SAL](image-url)
6.3.2 MD-SAL RESTCONF API

RESTCONF protocol, as said before, listens to port 8181 for HTTP requests. It operates a conceptual database modeled with YANG language. There are two types of access to the data stores via the controller: Config and Operational, containing data inserted by controller and other data inserted via network, respectively.

- Through this URL http://localhost:8181/apidoc/explorer/index.html RESTCONF can be accessed. The following figure shows the result:

Here, actions such as GET, PUT and DELETE actions are available.
6.3.3 How to add a flow

For a better understanding the same scenario used for AD-SAL explanation will be used (see section 6.2.1), with the difference that this time, REST APIs will be used:

6.3.3.1 Plugin Code

To add a flow to our switches it can be written in different languages, in this case it is written in XML. In this example the packets from the host3 will be dropped:
6.3.4 Test the Code

Before testing the code, the current flow tables installed in the switch can be checked through the following commands in Mininet:

```
$ pingall $ sh ovs-ofctl -O OpenFlow13 dump-flows s1
```

As shown in the Figure, a number of flows have installed when making the ping.

Now it is time to add the flow. The xml file is called “test4”. To do this, execute the following command:

```
$ curl -X PUT -d @test4 -H "Content-Type: application/xml" -H "Accept: application/xml" 
--user admin:admin http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/
openflow:1/table/0/flow/1
```

If we check now the flow tables and "pingall" we can see that our flow has been added and host3 does not have any communication with host1 and host2, the same results as the example
Figure 6.12: Flow tables before adding flow

Figure 6.13: PUT Flow

in AD-SAL writing application:

Figure 6.14: Results
In case that we wish GET or DELETE the added flow, execute in the console the following commands:

$ curl -X GET -d @test4 -H "Content-Type: application/xml" -H "Accept: application/xml"
   --user admin:admin http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1

$ curl -X DELETE -d @test4 -H "Content-Type: application/xml" -H "Accept: application/xml"
   --user admin:admin http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
Chapter 7

Conclusions and Future Work

In this thesis, the vision of future networks as well as its challenges has been displayed. Furthermore, the necessary information regarding the OpenDaylight Controller project is presented to guide the reader throughout the entire learning process of this subject.

In order to properly understand the different processes of writing an OpenDaylight application, important concepts such as Software Defined Networking (SDN), Network Function Virtualization (NFV), Cloud Computing or the OpenFlow protocol have been defined and studied. Also, Mininet has been presented and used as a network emulator.

Chapters 5 and 6, the core of the thesis, presented the OpenDaylight Controller project and the differing ways to work with it:

- Bundles that are installed in the OSGi framework, located beside the driver, run locally (AD-SAL and MD-SAL)
- REST instructions remotely executed.

A test case called "Solving a DoS Attack" is described, where a malicious host is launching a DoS attack to consume the SDN resources of the network. AD-SAL and REST API methods were used to create an application, which drops the packets coming from the attacker.

Having compared the three methods of writing an application for this controller, it is easy to see that the REST APIs method is far easier, faster and language independent. However, this method is unable to install reactive bundles and it does not provide the same flexibility and performance as the OSGi interfaces.
The two existing options available when programming bundles were explained. These are called API Driven SAL and Model Driven SAL. While AD-SAL is compiled through the code inside the interface, MD-SAL is more abstract and is compiled by applying models. This is a more innovative alternative and the community has already migrated from AD-SAL to MD-SAL. For that reason, MD-SAL is generally less documented, with fewer examples to help developers to understand and clarify its process.

At this moment in time, the OpenDaylight project is still under research. Potential future work might be to deploy and recreate the proposed examples into a physical network and study the MD-SAL architecture in further depth. This thesis can serve as a basis and an initiation for future OpenDaylight developers who wish to be part of the fascinating world of 5th generation networks.
Bibliography


[9] https://www.opennetworking.org/sdn-resources/sdn-definition


[16] https://www.opennetworking.org/Openflow


[19] https://wiki.opendaylight.org/view/Project_list


