How to control
a new network paradigm:
An overview of OpenDaylight SDN Platform

Tutors:
Prof. Ing. DAVIDE CAREGLIO
Prof. Ing. WALTER CERRONI
Prof. Ing. FRANCO CALLEGATI

Presentat per:
EDUARD REINA

Trimestre III
Any Acadèmic 2014/2015
‘Learn from yesterday,
live for today,
hope for tomorrow.

The important thing is not to stop questioning’

Albert Einstein.
Abstract

The need to increase efficiency in telecommunication is acquiring more and more relevance in today’s society, where the growing demand of the Internet services (from PCs and mobiles to even watches and cars) is over-flooding the current infrastructures. Due to stagnancy problems in traditional networks, the solution for this situation will not be centered in physically enlarging the present infrastructures.

The aim of this project is to describe and make understandable the new possibilities offered to the Network Telecommunication Operators, such as Network Functions Virtualization, Cloud Computing and Software Defined Networking paradigms, to face the aforementioned problem. The main features of this new approach are focused on the ability to design a dynamic and highly programmable network, moving the intelligence from the underlying systems to the network itself through a controller.

To assess the proper implications of this new approach, we will implement different dynamic and programmable networks that could simulate real scenarios and evaluate their performance contrasting the obtained results with the pragmatic theory. We will use a novel SDN (Software-Defined Network) controller (OpenDaylight), which will be exhaustively analyzed.

Different kind of networks will be solved through diverse OpenDaylight functionalities, either implementing the intelligence inside the controller (bundle), or
implementing it by an outside intelligent application (External Orchestrator) and finally sending it through OpenDaylight (by making OpenDaylight work as an interpreter/translator from its language to OpenFlow or another protocol language).

In order to gaze at a wide range of scenarios, we will use several OSI layers like L2 (for the connection ARP), L3 (network level, for the IP transport) or Application level (contacting with other programs to secure the network like a Data Packet Inspector (DPI)).

Summing up, these new and interesting technologies show an extraordinary capability to increase efficiency in Network communications, thus becoming an important asset to stir up the Network paradigm.
Abstract

La necessitat d’augmentar l’eficiència en les telecomunicacions està adquirint cada vegada més rellevància en la societat actual, on la creixent demanda dels serveis d’Internet (des d’ordinadors i mòbils fins a rellotges i cotxes) està saturant les infraestructures actuals. A causa de la dificultat d’evolució de les xarxes tradicionals, la solució no passa per l’ampliació de les presents infraestructures.

L’objectiu d’aquest projecte és descriure i fer comprensibles les noves possibilitats que s’ofereix als operadors de telecomunicació per fer front a l’esmentat problema. Aquestes passen per la implementació dels següents paradigmes: Network Functions Virtualization, Cloud Computing and Software Defined Networking. Les principals característiques d’aquest nou enfocament se centren en la capacitat de dissenyar una xarxa dinàmica i altament programable, movent la intel·ligència dels sistemes subjacents a la mateixa xarxa a través d’un controlador.

Per avaluar les propietats d’aquest nou disseny, implementem diverses xarxes dinàmiques i programables que simulen escenaris reals. Tot seguit, avaluem el seu rendiment contrastant els resultats obtinguts amb els teòrics. Per tot el procediment, utilitzem un controlador nou anomenat OpenDaylight, el qual és analitzat exhaustivament.

Es resolen una àmplia gamma de casuístiques en xarxes de telecomunicació a través de les moltes funcionalitats que ofereix OpenDaylight: ja sigui implementant la intel·ligència dins el controlador (bundle), o des de fora a través d’una aplicació.
independent (External Orchestrator) i finalment enviat la intel·ligència a través OpenDaylight (en aquest cas OpenDaylight treballa com a intèrpret / traductor del seu llenguatge al llenguatge OpenFlow o a qualsevol altre llenguatge de nivell de xarxa).

Per tal de contemplar un gran ventall d’escenaris, utilitzem diverses capes OSI, com per exemple a nivell L2 (per el protocol ARP), L3 (nivell de xarxa, per al transport IP) o el nivell d’aplicació (per a comunicar-nos amb altres programes (p. ex. Data Packet Inspector - DPI) i d’aquesta manera donar seguretat a la xarxa).

En resum, aquestes noves i interessants tecnologies demostren una capacitat extraordinària per augmentar l’eficiència en les comunicacions, convertint-se així en un actiu important per enriquir el Network Paradigm.
Abstract

La necesidad de aumentar la eficiencia en las telecomunicaciones está adquiriendo cada vez más relevancia en la sociedad actual, donde la creciente demanda de servicios de Internet (desde ordenadores y móviles hasta relojes y coches) está saturando las infraestructuras actuales. Debido a la dificultad de evolución de las redes tradicionales, la solución no pasa por la ampliación de las presentes infraestructuras.

El objetivo de este proyecto es describir y hacer comprensibles las nuevas posibilidades que se ofrecen a los operadores de telecomunicación para hacer frente a dicho problema. Estas pasan por la implementación de los siguientes paradigmas: Network Functions Virtualization, Cloud Computing and Software Defined Networking. Las principales características de este nuevo enfoque se centran en la capacidad de diseñar una red dinámica y altamente programable, moviendo la inteligencia desde los sistemas subyacentes a la misma red a través de un controlador.

Para evaluar las propiedades de este nuevo diseño, implementamos diversas redes dinámicas y programables que simulan escenarios reales. A continuación, evaluamos su rendimiento contrastando los resultados obtenidos con los teóricos. Para tal procedimiento, utilizamos un controlador nuevo llamado OpenDaylight, el cual es analizado exhaustivamente.

Se resuelven una amplia gama de casuísticas en redes de telecomunicación a través de las muchas funcionalidades que ofrece OpenDaylight: ya sea implemen-
tando la inteligencia dentro del controlador (bundle), o desde fuera a través de una aplicación independiente (External Orchestrator) y finalmente enviado la inteligencia a través OpenDaylight (en este caso OpenDaylight trabaja como intérprete / traductor de su lenguaje al lenguaje Openflows o cualquier otro lenguaje de nivel de red).

Para contemplar un gran abanico de escenarios, utilizamos varias capas OSI, como por ejemplo a nivel L2 (para el protocolo ARP), L3 (nivel de red, para el transporte IP) o el nivel de aplicación (para comunicar -nos con otros programas (p. ej. Data Packet Inspector - DPI) y de esta manera dar seguridad a la red).

En resumen, estas nuevas e interesantes tecnologías demuestran una capacidad extraordinaria para aumentar la eficiencia de las telecomunicaciones, convirtiéndose así en un activo importante para enriquecer el Network Paradigm.
# Contents

Abstract i

1 Introduction 1

1.1 Background: From the carrier of today, to the carrier of tomorrow 2
1.2 Objectives .................................................. 3
1.3 Explanation of the thesis structure .................................. 4

2 Technologies and tools for a new network paradigm 7

2.1 Network Function Virtualization .................................. 7
2.2 Software Defined Networking .................................... 8
2.3 Cloud Computing ............................................. 11
2.4 OpenFlow .................................................. 12
2.5 Mininet ..................................................... 13
2.6 OpenDaylight ................................................ 14

3 OpenDaylight 15

3.1 Introduction to the controller ..................................... 15
3.1.1 History ................................................ 16
3.1.2 Technology Overview .................................... 16
3.1.3 Model View Controller (MVC) platform ...................... 17
3.1.4 Fundamental Software Tools ............................... 17
3.2 Advantages in front of other controllers ....................... 18
3.3 The Structure .................................................. 20
3.4 Two different architectures (ADSAL and MDSAL) ............... 22
3.5 Packet path in OpenDaylight managing OpenFlow devices ..... 27
  3.5.1 In ADSAL architecture: .................................. 27
  3.5.2 In MDSAL architecture: ................................. 29

4 Inside Programming: Developing OSGi bundles 35
  4.1 An ADSAL Learning Switch: .................................. 35
  4.2 Extension to a Multi Switch Network topology ................... 52
  4.3 Another ADSAL Bundle: Specific for a concrete Topology .... 53
  4.4 An MDSAL Learning Switch .................................. 55

5 Outside Programming: common Rest API 57
  5.1 Introduction .................................................... 57
  5.2 Current API .................................................... 58
    5.2.1 REST ....................................................... 59
    5.2.2 JSON ....................................................... 59
    5.2.3 XML ......................................................... 59
    5.2.4 HTTP ....................................................... 60
  5.3 Possible solution to solve reactive flow programming ......... 61
  5.4 Installing and reading Flows ................................ 63
    5.4.1 Settings ................................................... 65
    5.4.2 Postman REST-Client .................................... 66
    5.4.3 Java Code ................................................ 75

6 The External Orchestrator: Intelligent Outside App 81
  6.1 The scenario .................................................. 83
  6.2 Reactive and Proactive capability ............................... 84
ABSTRACT

6.3 Three kinds of flows .............................................. 85
   6.3.1 L2 flows (ARP) .............................................. 85
   6.3.2 L3 Initiation flows ......................................... 86
   6.3.3 L3 definitive flows ........................................ 86
6.4 The Structure ...................................................... 87
   6.4.1 GUI User Interface (1) ..................................... 88
   6.4.2 Settings ...................................................... 88
   6.4.3 Main External Orchestrator ................................. 89
   6.4.4 Flow Initializer ............................................ 89
   6.4.5 Rest Interface JSON Creator ............................... 93
   6.4.6 Rest Interface Sender ..................................... 94
   6.4.7 Stock Socket Quote Server ................................ 94
   6.4.8 Chain Reader ............................................... 95
   6.4.9 Full Matrix ................................................ 95
   6.4.10 Flow Repetition Control (13) ............................ 96
   6.4.11 Flow Model ............................................... 97
   6.4.12 Rest Interface Receiver .................................. 98
   6.4.13 Rest Interface JSON Analyzer ............................ 99
   6.4.14 Installing a new L3 definitive flow and waiting for the next one (19-23) ......................... 102

A Code Developed ......................................................... 109
   A.1 Mininet Topology (Python script): Two directly connected switches plus a host for each switch ......................... 110
   A.2 Mininet Topology (Python script): Specific Loop Topology .... 111
   A.3 ADSAL Hub .................................................... 112
   A.4 ADSAL Bundle Activator ...................................... 116
   A.5 ADSAL Learning SINGLE Switch ............................... 118
List of Figures

1.1 Traditional networks against new approaches .................. 3

2.1 Structure of a SDN network .................................. 11

2.2 Diagram of a Cloud Computing Scenario ...................... 12

3.1 OpenDaylight Founding Members [30] .......................... 19

3.2 OpenDaylight modular structure platform [25] .................. 20

3.3 Connection between OSGi bundles ............................ 21

3.4 Comparison between ADSAL and MDSAL architectures ........ 22

3.5 Packet path in ADSAL managing OpenFlow devices [31] ...... 27

3.6 Packet path in MDSAL managing OpenFlow devices .......... 29

4.1 Diagram with all the steps to create an ADSAL bundle .......... 37

4.2 Steps to process a packet in the Packet Handler of an ADSAL bundle 43

4.3 DLUX, the Graphical User Interface of OpenDaylight .......... 50

4.4 An specific possible real scenario to develop an OSGi bundle ... 54
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Projects and its dependencies in OpenDaylight Lithium</td>
</tr>
<tr>
<td>5.1</td>
<td>A possible solution to implement reactive flow programming through REST</td>
</tr>
<tr>
<td>5.2</td>
<td>Steps to install and read flows from outside by REST through OpenDaylight</td>
</tr>
<tr>
<td>5.3</td>
<td>Postman REST client screenshot</td>
</tr>
<tr>
<td>5.4</td>
<td>API-DOC of OpenDaylight</td>
</tr>
<tr>
<td>6.1</td>
<td>Hypothetical scenario controlled by an External Orchestrator</td>
</tr>
<tr>
<td>6.2</td>
<td>Our External Orchestrator structure</td>
</tr>
<tr>
<td>6.3</td>
<td>GUI User Interface of the External Orchestrator screenshot</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Over the last years, the hottest topics in networking have been Software Defined Networking (SDN), Network Virtualization (NV) and Cloud Computing. It is not surprising that nowadays there is urgent need to increase the efficiency in the telecommunication networks due to the growing demand of internet. Each day more and more devices are using this technology: cars, glasses, watches, houses and of course mobiles which are indispensable for almost the entire population.

Not only is the increasing number of devices an issue to be solved, but also and even more important, the increasing data that they are managing: everybody wants elevated number of Gigabytes of data in their mobile contracts to send photographs, videos, to use social networks, to surf the web and this is leading to the need to support more and more weight...

Thus, this demand is over-flooding the current services and infrastructures. To face it one might think that the solution comes by enlarging the present infrastructures (if there are too many cars, let’s make more motorways), however this solution represent a lot of troubles.
1.1 Background: From the carrier of today, to the carrier of tomorrow

Traditional Data Network: In the traditional network approach, the major part of the network functionality is implemented in the devices like switches or routers, and in dedicated hardware. This scheme implies some difficulties in network functionality making it to evolve slowly. Another quandary is the user usability: the hardware networks functionality is under the control of the provider of the appliance making it quite static.

In addition, networking organizations are under increasing pressure to be as efficient and agile as possible with the traditional approach. One source of this pressure arise from the widespread adoption of Cloud Computing. As part of this server virtualization, virtual machines (VMs) are dynamically moved between servers in a matter of seconds or minutes. However, due to its hardware implementation, if the movement of a VM crosses a Layer 3 boundary, it can take days or weeks to reconfigure the network to support the VM in its new location, with its consequently error prone.

Thus, the emergence of Cloud Computing is gradually transforming the up to now traditional hardware-centric data network to a software-based network functionality. This means that functions such as encryption/decryption and the processing of TCP flows, which were previously performed in hardware designed specifically for those functions, are now driven largely, by the need for agility increase, to software running on a general purpose server or on a VM.

In a nutshell, the traditional network architectures are ill-suited to make the requirements of the today’s users and enterprises. Also, a wide range of new opportunities bound to the networking virtualization are opened, such as Software Defined Networking (SDN) and Network Virtualization (NV). With this new approach, the network can improve in scale functionality and easily deployment, performing in traffic engineering with an end-to-end view and better utilization of
the resources. Furthermore, the price of maintenance will be reduced since most of the control will be devolved by software platforms. Lastly, network functionality evolve more rapidly when it is based on software development lifecycle and it enables applications to dynamically request services from the network. Not to mention, the more effective security functionality and the reduced complexity.

Figure 1.1: Traditional Network in the left. In the right, new kind of networks based in Cloud Computing, SDN and NFV

1.2 Objectives

The aim of this project is to describe and make understandable the new paradigms offered to the Network Telecommunication Operators –such as Network Functions Virtualization, Cloud Computing and Software Defined Networking– to face the problems caused by the traditional networks.

The main features of these new approach are focused on the ability to design a dynamic and highly programmable network, moving the intelligence from the underlying systems to the network itself through a controller. This means that the data plane is decoupled from the control plane. This control plane is managed by a controller that increase incredibly the user usability of the network: if a user/client want to develop his own network with his own conditions and topology, will be able to do it through this new tool, the controller.
1. Introduction

Thus, our objective is to be this user/client and enquire into one of these controllers, OpenDaylight. There are some kinds of different controllers like POX or RYU. However we will use the novel OpenDaylight meant to be one of the most used thanks to some advantages in front of others: like having a REST interface to communicate with outside applications.

Indeed, we will try to investigate as many functionalities as possible that this controller offers and we will try to use, manipulate and program them. We will also develop new functionalities. Everything in order to simulate and solve possible real scenarios that a user (enterprise, network provider...) should had to deploy. Finally, we will evaluate the behavior of the implemented scenarios.

1.3 Explanation of the thesis structure

In order to follow the aforementioned objectives (Objectives page XX), we will proceed with the following steps divided in three main blocks: the theoretical part, the practical part and the conclusions:

The dissertation begins with an Abstract and an Introduction to orient in the concerned scenario. It is explained the problematic of the traditional hardware based networks and the importance to migrate them into an innovative software centred approach using the following paradigms: Network Function Virtualization, Software Defined Networking and Cloud Computing.

In the following section we set the Objectives: briefly we try to explain how we deal to acquire advanced knowledge in this new paradigms. Also, how to proceed after the obtainment of this knowledge, to use it for the simulation and solution of possible real scenarios using this technology.

In the second chapter, Technologies and tools for a new network paradigm, begins the theoretical block where we explain the technologies and tools used
during the thesis to imitate networks and to manage its behavior.

Once introduced the main instruments that we will use, in the third chapter we focalize in **OpenDaylight**. To implement networks using the new paradigms we need an essential device, the controller. OpenDaylight is a new controller and the one used in this work. In order to be able to manage networks with it later, in this section we investigate deeply its features.

At this point the theoretical part is over and we begin to apply them in **practical cases**.

The first practical case that we face is in the fourth chapter, **Developing OSGi Bundles for OpenDaylight**, when we develop new services called Bundles to solve particular scenarios. This services must be implemented inside OpenDaylight, thing that differentiate this section with the following once.

In the fifth chapter, **the Model-Generated common Rest API**, we also implement services to manage networks, but in this case we develop them from outside the controller. In this case, OpenDaylight will translate from the protocol used outside to the protocol used by the network devices, in our case OpenFlow.

Bounded with the previous section, in the chapter six we propose an **Outside Orchestrator: an Intelligent External Application to manage networks**. This one is connected with other external programs in order to control different scenarios and use OpenDaylight to transmit this intelligence to the network devices.
1. Introduction
Chapter 2

Technologies and tools for a new network paradigm

To carry on with this dissertation we need a wide range of instruments. In this section we see a brief overview of the most important ones in order to be able to know them when we need.

It has to be highlighted that some of them are the technologies (NFV, SDN, Cloud Computing, OpenFlow) used to carry out a new network paradigm, and the other ones are the tools utilized to simulate the scenarios supporting this technologies (Mininet, OpenDaylight).

2.1 Network Function Virtualization

The Network Function Virtualization (NFV) concept has appeared recently. It was in a conference about Software Defined Networking and OpenFlow in October 2012, when it first appeared a paper about this paradigm written by a telecommunication industry group belonging to European Telecommunications Standards
Institute (ETSI) (reference!).

In the mentioned paper, the authors expose the aforementioned troubles of implementing networks in the traditional approach due to its hardware-based composition. Straightaway they introduce a proposal to replace the hardware middleboxes and the vendor dependency mechanisms leveraging virtualization of operating systems, thus implementing the middleboxes using general-purpose hardware, in order to be dynamically reprogrammable and inexpensive.

In a nutshell, Network-function virtualization is a network architecture concept that proposes to virtualize entire classes of network node functions into general-purpose reprogrammable hardware that may be connected, or chained, to create communication services. This virtualization allows for agile placement of networking services when and where they are needed, also reducing the costs.

### 2.2 Software Defined Networking

As previously explained, the computing, storage and connectivity services are changing from a static hardware structure to a software-centered virtualization. This is happening under the cloud paradigm, where the network functionality can be virtualized and carried out in a generic hardware. And this transformation is starting to become relevant in the field of networks.

Having said that, seems clear to focus the research on creating a new paradigm of protocols to carry out this virtualization. The Open Networking Foundation (ONF) is leading the development and standardization of SDN. According to the ONF, "Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted
for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions.

According to the ONF, the SDN architecture is:

- **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.

- **Business Applications**: This refers to applications that are directly consumable by end users. Possibilities include video conferencing, supply chain management and customer relationship management.

- **Network and Security Services**: This refers to functionality that enables business applications to perform efficiently and securely. Possibilities include a wide range of L4 - L7 functionality including ADCs, WOCs and security capabilities such as firewalls, IDS/IPS and DDoS protection.
• **Pure SDN Switch:** In a pure SDN switch, all of the control functions of a traditional switch (i.e., routing protocols that are used to build forwarding information bases) are run in the central controller. The functionality in the switch is restricted entirely to the data plane.

• **Hybrid Switch:** In a hybrid switch, SDN technologies and traditional switching protocols run simultaneously. A network manager can configure the SDN controller to discover and control certain traffic flows while traditional, distributed networking protocols continue to direct the rest of the traffic on the network.

• **Hybrid Network:** A hybrid network is a network in which traditional switches and SDN switches, whether they are pure SDN switches or hybrid switches, operate in the same environment.

• **Northbound API:** The northbound API is the API that enables communications between the control layer and the business application layer. There is currently not a standards-based northbound API.

• **Southbound API:** The southbound API is the API that enables communications between the control layer and the infrastructure layer. Protocols that can enable this communications include OpenFlow, the extensible messaging and presence protocol (XMPP) and the network configuration protocol.

To sum up, with SDN paradigm the data plane is decoupled of the control plane: the intelligence of the network and its state are centralized in an external dispositive called controller. The structure underlying the transport plane is abstracted from the application and network services. With this new concept, the usability is highly increased allowing the operators, enterprises or other users to get a lot of programmability, automatization and network control. This allows to quickly adapt each network to its changing environment and new needs.
2.3 Cloud Computing

Cloud computing is a model that relies on sharing computing resources in a shared configurable pool instead of having local servers or personal devices to handle applications. This paradigm provides users and enterprises with various capabilities to store and process their data in third-party data centers. Its philosophy falls on sharing resources to achieve coherence and economies of scale, similar to a utility (like the electricity grid) over a network.

Figure 2.1: Structure of a SDN network [18]
2.4 OpenFlow

OpenFlow is the first standard communications interface defined between the control and the forwarding layers of an SDN architecture. It is developed by the OpenFlow Network Foundation (ONF) and allows the controller to have direct access to the forwarding plane of network devices such as switches and routers, both physical and virtual. It also give the possibility to manipulate this forwarding plane. To sum up, OpenFlow is the standard communications interface that will allow us to connect the network devices (switches, routers...) with the southern layer of the controller (OpenDaylight in our case) that is called Service Abstraction Layer (SAL).
How does it work?

In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow Switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow Switch and Controller communicate via the OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modify-forwarding-table or get-stats.

The data path of an OpenFlow Switch presents a clean flow table abstraction: each flow table entry contains a set of packet fields to match (e.g. inPort, @IP source, @IP destination, type of the packet,...) and an action (e.g. outPort, modify-field, drop...). When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directing the switch on how to forward similar packets in the future.

2.5 Mininet

Mininet is the network emulator used in this research. It can run a collection of hosts, switches, routers and links on a single Linux kernel. Despite its lightweight virtualization, it is able to make a single system look like a complete network, running the same kernel, system, and user code.

The user is who configures how the network is acting with Python scripts and commands (Appendix A.1, A.2, A.8). Just to introduce how Mininet works (further explain in future chapters), a Mininet host behaves just like a real machine so is possible to access by ssh into it (if we have start up sshd and bridge the network to our host) and run arbitrary programs (including anything that is installed on
the underlying Linux system). This option will be very useful in this project to analyze the behavior of the controller sending different kind of packets.

Shortly, Mininet’s virtual hosts, switches, links and controllers are very close to discrete hardware elements for the most part of their behavior. However, they are created using software. It is essential to simulate a scenario that we would eventually implement. For this reason electrical engineering simulate circuits with PSPICE, architectures design their plans with AUTOCAD and we simulate the networks with Mininet.

2.6 OpenDaylight

OpenDaylight is the network controller that we use in this dissertation to control networks using the new paradigms. According to its manual, it is an open platform for network programmability to enable SDN and create a solid foundation for NFV for networks at any size and scale.

It has a highly available, modular, extensible, scalable and multi-protocol infrastructure built for SDN deployments on modern heterogeneous multi-vendor networks. OpenDaylight also provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and south-bound protocols.

To completely understand this brief introduction to OpenDaylight platform, go to the following Chapter (Chapter 3) where a complete overview is presented.
Chapter 3

OpenDaylight

As aforementioned, OpenDaylight is the controller that we use in this dissertation to control networks using the new paradigms of Network Function Virtualization, Software Defined Networking and Cloud Computing. This section is an accurate and theoretical overview of this controller, in order to further use it to program the control of different network scenarios.

3.1 Introduction to the controller

The OpenDaylight Project is a collaborative open source project hosted by The Linux Foundation. According to its partners, the goal of the project is to accelerate the adoption of Software Defined Networking and create a solid foundation for Network Functions Virtualization.
3.1.1 History

Some months after the Software Defined Networking site “SDN Central” broke news of an industry coalition forming around SDN; on April 8, 2013, The Linux Foundation, announced the founding of the OpenDaylight Project. This platform was exposed as a community-led and industry-supported open source framework to accelerate adoption, foster new innovation and create a more open and transparent approach to Software Defined Networking and Network Functions Virtualization.

From that point on, three major versions have been released: Hydrogen (February 2014), Helium (September 2014) and the current Lithium (June 2015). During this period, an increasing number of companies have given support to the project: since the foundational and very powerful ones such as Cisco, Intel, Microsoft... to other newer names like Huawei or Lenovo, until reach the astonishing number of almost 50 influential brands.

3.1.2 Technology Overview

OpenDaylight is a modular platform written in Java with most modules (bundles) reusing some common services and interfaces (Service Abstraction Layer, SAL).

Each module is a service offered by the controller and it is developed under a multi-vendor sub-project.1 Following the idea of SDN, each user can manipulate these already implemented bundles and also can develop his own one in order to control his particular network. This gives many possibilities to personalize the network to the user.

The idea of building applications on the OpenDaylight platform with this bundles structure is to leverage functionality in other platform bundles, each of which export important services through Java interfaces. The major part of these services are built in a provider-consumer model over an adaptation layer called SAL.
Thus, the SAL is layer used to establish the connection between everybody, not only between bundles, but also between inside the controller (bundles) and the network devices (nodes, switches...), and even with the external applications.

### 3.1.3 Model View Controller (MVC) platform

OpenDaylight is a Model-View-Control platform. This means that the application is divided in three interconnected parts, in order to separate internal representations of information from the ways that information is presented to or accepted from the user:

- **Model → YANG**: Model for data, RPC and notifications. For example a model of a flow (Chapter 5.4, Appendix B.3, B.4). Used in MDSAL architecture (Chapters 3.4, 3.5.2, 4.4, 5).

- **View → REST API**: View self-generated and accessible through Northbound (ADSAL, Chapters 3.4, 3.5.1, 4) or RESTconf (MDSAL, Chapter 5). It is our user interface to see the information (e.g. a flow).

- **Control → Java Implemented Code**: To handle data changes, notifications and RPC call backs.

### 3.1.4 Fundamental Software Tools

OpenDayLight uses the following software tools/paradigms. It is important to become familiar with them.

- **Java interfaces**: The already mentioned Java Interfaces are used for event listening, specifications and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.
Maven: It is a software tool used for the management and construction of Java projects. OpenDayLight uses Maven for easier build automation. Maven uses pom.xml (Project Object Model for this bundle) to script the dependencies between bundles and also to describe what bundles to load on start (Chapter 4.1.3, Appendix B.5).

OSGi: This framework in the backend of OpenDayLight allows dynamically loading bundles and packaged Jar files, and binding bundles together for information exchange.

Karaf: Karaf is a small OSGi based runtime which provides a lightweight container for loading different modules. It will be available after the first release, so for Helium and Lithium releases.

3.2 Advantages in front of other controllers

Different kind of controllers can be used in order to deploy networks in the Software Defined Networking paradigm: like POX, RYU or ONOS. Among the available ones, there are some reasons to choose OpenDaylight:

- **Open Source:** As an open source platform, OpenDaylight platform provide a universal access via a free license to the platform, and universal redistribution of that platform including subsequent improvements to it by anyone. According to this:
  
  - The Linux Foundation manages it.
  - Anyone interested is free to collaborate.
  - We have free access to the platform.
  - We can modify its services and behavior by changing its prebuilt modules of implementing new ones.
• **Industry support:** The industrial support of this platform encompasses the major part of the most powerful companies in the IT field. Some very important firms like Cisco, Intel, Microsoft... among many others, are part of the OpenDaylight members.

• **Novel functionalities:** As a new platform, OpenDaylight implements some innovative functionalities respect other controllers. For instance the possibility of contacting with external applications that are not strictly connected with the network. This functionality is very useful and it is widely explained during the thesis, reaching the point that Chapters 5 and 6 are focalized on this topic.

• **Running in a personal Java Virtual Machine (JVM):** The main advantage is the portability that it offers. Running the platform in its own JVM proportionate the possibility to use and develop OpenDaylight on any hardware that support Java.
3. OpenDaylight

3.3 The Structure

The following picture is offered by the OpenDaylight official site and shows the structure of the platform and the tools or devices, which it can contact with.

Figure 3.2: OpenDaylight modular structure platform [25]

To comprehend the full platform overview, it is important to understand the above diagram (figure XX):

- **OpenFlow Enabled Devices**: This is the network infrastructure managed via OpenDaylight. In this case OpenFlow plug-in.

- **Protocol Plugins**: OpenDaylight supports these protocols for managing your network devices. One of the plugins is OpenFlow.
3.3 The Structure

- **Service Abstraction Layer (SAL):** This layer does all the plumbing between the applications and the underlying plugins.

- **Controller Platform:** These are the applications that come pre-bundled with OpenDaylight to manage the network. It is possible to write our own bundle.

- **Bundles:** They are each of the green small boxes inside the Controller Platform. The figure below (figure XX) shows their interconnections. It consists of a number of OSGi bundles that bundle together Java classes, resources, and a manifest file. Some of them are coming from the OpenDaylight project like the SAL (Service Abstraction Layer) bundle. Bundles are executed atop the OSGi Framework. The interesting thing about OSGi is that bundles can be installed and removed at runtime, so you do not have to stop the SDN controller to add or modify control logic.

![Diagram of OSGi bundles and connections](image)

Figure 3.3: Connection between OSGi bundles [31]

OSGi bundles are offering services that can be called by other OSGi components as for example the Data Packet Service (interface IDataPacketService) to decode data packets.
• **Network Applications**: These are applications with leverage REST NBI of OpenDaylight to build intelligence.

### 3.4 Two different architectures (ADSAL and MD-SAL)

Aforementioned the Service Abstraction Layer, or SAL, is nothing more than a pipe that connects the Protocol Plugins (see graphic XX) with the bundles and the REST APIs. Thus, the SAL API are the contract that the Protocol Plugins and the NFS (Application on the top of SAL layer: bundles, external applications...) sign, in order to be able to communicate to each other. Also it is used to talk between bundles themselves or bundles and REST API.

At this point, is important to mention the existence of two different kinds of SAL APIs, the API Driven SAL and the Model Driven SAL.

![Comparison between ADSAL and MDSAL architectures](image)

**Figure 3.4: Comparison between ADSAL and MDSAL architectures [25]**
To enhance this knowledge, the table below lists the main differences between each approach:

<table>
<thead>
<tr>
<th>API Driven SAL (AP-SAL)</th>
<th>Model Driven SAL (MD-SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The SAL APIs, request routing between consumers and providers. The consumers are the</td>
<td>It also requests routing between consumers and providers, but in MDSAL case is defined</td>
</tr>
<tr>
<td>NFS (Applications on the top of the SAL layer). The providers are the Protocol-plugins,</td>
<td>from models. The data adaptation is provided by an internal adaptation plugin. A new</td>
</tr>
<tr>
<td>which provide the Southbound services. This request routing, together with the data</td>
<td>bundle can be started while the controller is running.</td>
</tr>
<tr>
<td>adaptation, are all statically defined at compile/build time.</td>
<td></td>
</tr>
<tr>
<td>From an external application, if we want to contact with the controller it will be</td>
<td>Nevertheless, in the MD-SAL the plugins from both layers (NB and SB), use the same</td>
</tr>
<tr>
<td>necessary the Northbound API (see Chapter 5.1). The same happens with the Southbound</td>
<td>API generated from a model. This model, as for example a flow (Chapter 5.4, Appendix</td>
</tr>
<tr>
<td>(SB) to contact the Protocol Plugins. So NB and SB are separated even for functions or</td>
<td>B.3, B.4), can be used by anybody for its own purposes. For example, it can be used by</td>
</tr>
<tr>
<td>services that are mapped 1:1 between SB Plugins and NB Plugins.</td>
<td>an external application through the NB (Chapter 6 figure 6.1), by Protocol Plugins (e.g</td>
</tr>
<tr>
<td></td>
<td>OpenFlow) through the SB, or also by other bundles inside the Controller Platform.</td>
</tr>
<tr>
<td>As before said, NB and SB are completely separated. For these reason there is a</td>
<td>In this case, MD SAL provides a common REST API to access data and functions defined in</td>
</tr>
<tr>
<td>dedicated REST API for each of the bounds, NB and SB.</td>
<td>models.</td>
</tr>
<tr>
<td></td>
<td></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. This storage is in the core of the platform and logically centralized data store that keeps relevant state in two buckets: 1) config data store, 2) operational data store. Thus, here is where the model will be saved. An application, an external user, another bundle or the Protocol plugins can ask, modify or add information here, for example a flow (Chapter 5.4, Appendix B.2, B.3, B.4).</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>The AD-SAL provides request routing (selects a SB plug-in based on service type) and optionally provides service adaptation if is needed. For example, by some interfaces like IlistenDataPacket (figure 3.3) we can get the packet-in that has been sent by OpenFlow through the SB API and process it with some specific java classes and methods already developed by ODL. Then can be sent to bundles, to an external application or to OpenFlow again.</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. For example, when we ask though a REST-call a certain flow installed in a switch, we obtain the model of each flow. Then, the process of this information has to be made by ourselves, or we can ask help from some plugins already developed.</td>
</tr>
</tbody>
</table>
### 3.4 Two different architectures (ADSAL and MDSAL)

<table>
<thead>
<tr>
<th>ADSAL</th>
<th>MDSAL</th>
</tr>
</thead>
<tbody>
<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 it is not limited to flow-capable device and service models only.</td>
</tr>
<tr>
<td>More useful for a reactive flow scenario. Reactive flow instantiation: When a new flow comes into the switch, the OpenFlow agent SW on the switch, does a lookup in the flow tables. If no match for the flow is found, the switch creates an OF packet-in and fires it off to the controller for instructions. Reactive mode reacts to traffic, consults the OpenFlow controller and creates a rule in the flow table based on the instruction.</td>
<td>More useful for proactive flow scenario. Proactive flow instantiation: Rather than reacting to a packet, an OpenFlow controller could populate the flow tables ahead of time for all traffic matches that could come into the switch. This scenario is very used when it is known a priori the network structure and how it has to work, so the flows can be sent before the network begins to work. I take this opportunity to remind that this is a common situation in SDN, when the user want to give a solution to his own particular network.</td>
</tr>
</tbody>
</table>

The conclusions to follow up with the project are:

- ADSAL approach is more useful when we want to control/program a network where we are allowed to access directly to the controller. It is also important to mention that the controller should not need to interchange much information between an external program through the NB, to OpenFlow through the SB, since this approach does not provide a common REST API.

  In these favorable cases, the advantages of this architecture fall on the usability of the service adaptation which is well developed and with a wide range of Java classes and methods to process a packet-in.
• In the other hand, if we cannot access physically to program the controller or we need to access the controller by an external application that needs to often contact OpenFlow, the best way is MD SAL approach. The reason is that MD-SAL provides a common infrastructure where data and functions defined in models can be accessed by means of a common REST API. So, installing an OF flow directly through the NB is quite simpler.
3.5 Packet path in OpenDaylight managing OpenFlow devices

Once we know with more detail the OpenDaylight architecture, let’s see which elements are involved in the installation of a flow in the both possible SAL approaches:

3.5.1 In ADSAL architecture:

In ADSAL the packet path begins from an OpenFlow enabled network device and goes to the controller. Then, the controller processes it and it comes back to the network device.

![Packet path in ADSAL managing OpenFlow devices](image-url)
In the picture it is shown how a packet flows between an OpenFlow enabled network device to the controller and which interfaces are being used. It is also explained which the return back path of the packet is:

1. A packet arriving at a network device, like an OpenFlow switch, is sent to the appropriate protocol plugin of OpenDaylight which is managing the switch.

2. IPluginOutDataPacketService: The plugin (in our case OpenFlow Plug-in) will parse the packet, generating an event for Service Abstraction Layer (SAL) for further processing.

3. IListenDataPacket: SAL will dispatch the packet to all the modules listening for DataPacket. If the first packet is just an ARP Request, SAL will send it to the ARP Handler. However if it is another kind of packet, it will be processed by its pertinent bundle. This configuration needs to be set accurately (See Chapter 4.1.8) otherwise some packets will not reach their destination/goal. This configuration and also all these bundles can be changed. Also other bundles can be created following the user necessities. In this thesis it will be further shown how to create or modify this bundles.

4. IDataPacketService: The Application/Module that has been commented in (3) is in charge of the packet processing in accordance with its own needs, so it is the user who can actually program it. Finally a PACKET OUT is sent using IdataPacketService interface. For example, after processing the packet, the PACKET OUT can be a rule of where the OpenFlow switch has to send the packet and also install in this device a new flow.

5. IPluginInDataPacketService: SAL receives the DataPacket and dispatches it to the modules listening for plug-in DataPackets. In this case OpenFlow plugin.

6. OpenFlow plugin then sends the packet back to the device from where the packet was originated. If we follow the previous example, once the switch receive the DataPacket, it will know where to send the packet previously
3.5 Packet path in OpenDaylight managing OpenFlow devices

3.5.2 In MDSAL architecture:

ADSAL is not the only way to insert flows. In the following figure it is shown a scenario where an external application adds a flow by means of RESTconf API of the controller. Whereas in the previous example we were dealing with a reactive approach, in this case it is a proactive scenario.

1. When the controller is started with its corresponding plugins, the followings registrations are performed:
   - The Flow Programmer Service registers with the MD SAL for Flow configuration data notifications.

Figure 3.6: Packet path in MDSAL managing OpenFlow devices [21] sent to the controller. In addition, it will install a new flow in order to flow directly the upcoming packets with the same characteristics.
3. OpenDaylight

- The registration between the OF Plugin and the MDSAL between a Remote Procedure Call (RPC) to establish a connection between both platforms. Note that the RPC is defined in the OF Plugin model, and that the API is generated during build time.

2. A client application that would be running anywhere establish a REST connection with the REST API of the OpenDaylight controller sending a flow add request. As has already been said, in AD-SAL there is a dedicated NB REST API on top of the Flow Programming Service. The MD-SAL provides a common infrastructure where data and functions defined in models can be accessed by means of a common REST API. The client application provides all parameters for the flow in the REST call.

3. Data from the 'Add Flow' request is deserialized, and a new flow is created in the Flow Service configuration data tree. (Note that in this example, the configuration and operational data trees are separated; this may be different for other services). Note also that the REST call returns success to the caller as soon as the flow data is written to the configuration data tree.

4. The MD-SAL generates a 'data changed' notification to the Flow Programmer Service since this one is registered to receive notifications for data changes in the Flow Service data tree.

5. The Flow Programmer Service reads the newly added flow, and performs a flow add operation.

6. At this point the Flow Programmer Service tells the OF Plugin to add the flow in the appropriate switch. The Flow Programmer Service uses the OF Plugin generated API to create the RPC input parameter, called DTO, for the 'AddFlow' RPC of the OF Plugin. So basically the Flow Programmer Service communicates the OF Plugin API that it wants to add a new flow.

7. The Flow Programmer Service gets the service instance (actually, a proxy), and invokes the 'AddFlow' RPC on the service. The MD-SAL will route
the request to the appropriate OF Plugin (which implements the requested RPC).

8. The 'AddFlow' RPC request is routed to the OF Plugin, and the implementation method of the 'AddFlow' RPC is invoked.

9. The 'AddFlow' RPC implementation uses the OF Plugin API to read values from the DTO of the RPC input parameter. (Note that the implementation will use the getter methods of the DTO generated from the yang model of the RPC to read the values from the received DTO.)

10. The 'AddFlow' RPC is further processed (pretty much the same as in the AD-SAL) and at some point, the corresponding flowmod is sent to the corresponding switch.
3. OpenDaylight
Annotations and Reflections

Once arrive here, the theoretical part of this dissertation is over and it is the moment to emphasize some topics in order to go on with the practical block.

The main improvement that OpenDaylight offers is the possibility of controlling a network by three different ways:

- We can control our scenario by implementing a bundle inside the controller: This module will probably be developed in the ADSAL architecture due to the facilities this layer give when we program inside the platform.

- In some occasions the control of the network have to be implemented outside the controller for different causes, for example:
  - Standard reasons: In order to follow standards and not change radically the default composition of OpenDaylight platform.
  - Portability reasons.
  - Different programming language: If we want to program inside the controller we are forced to use Java. If we have already implemented the program in another language or we have the intention to do it, we have to develop it externally.
  - Intelligence of the network already implemented externally.
  - Intention to contact frequently with other outside applications that provides some information.
In these situations, MDSAL is the most intelligent choice as it provides a common REST API from outside the controller to the network devices.

- Finally, we can develop a hybrid solution in which we program the intelligence of the network externally but also we take advantage of some bundles already implemented in OpenDaylight.

From now on, the goal is to simulate different kind of possible real scenarios and solve them programming the intelligence making use of one of these different ways, taking care which one is more appropriate for the network at hand.

Thus, let’s continue with the practical block.
Chapter 4

Inside Programming: Developing OSGi bundles

When we chose to program the intelligence of the network inside the controller, the best choice is to develop an OSGi bundle to carry out this job. In the present section we deploy some different bundles to control diverse scenarios.

4.1 An ADSAL Learning Switch:

The most important characteristic about OSGI bundles, in contrast to the REST interface (further analyzed in chapter 5), is that it is able to receive packet-in events, which are triggered when a packet without matching flow table entry arrives at a switch. Therefore, to do reactive flow programming, OSGi components are the right way to proceed in OpenDaylight.

In order to receive these packet-in events, the bundle has to offer a service implementing the IListenDataPacket interface. Whenever an OpenFlow packet-in event arrives at the controller, the SAL invokes the components that implement
the IListenDataPacket interface, among them our bundle.

In the first bundle that we implement, we develop a L2 Learning Switch for a topology with only one switch. A possible solution for this situation it is already implemented in the L2Switch bundle provided by default in the OpenDaylight controller. However, we develop from scratch our version in order to become acquainted with the bundle creation and create more complex modules a posteriori (Chapter 4.2, 4.3, 4.4).

This module implements two functionalities:

- An ARP flooding service for either the ARP packets or the packets that the switch do not know the destination MAC address.

- An L2 packet deliver service and flow installing service: when a packet arrives to the controller, it associates the source MAC address with the port where the packet has been delivered. Then, when a later packet arrives to the controller with the destination MAC address equals to the previous packet source MAC address, the controller now knows to which port has to be sent. This rules are sent by the controller installing a new flow. In this particular case, it is the same procedure that a traditional switch does to fill its route table, but done through the controller. Afterwards we will see that using the controller device we will be able to make many more things rather than with a traditional switch.

To deal with this first bundle implementation, we design a diagram with all the steps explained in minute detail:
4.1 An ADSAL Learning Switch:

Figure 4.1: Diagram with all the steps to create an ADSAL bundle
1. **Install OpenDaylight:** Aforementioned, there are different releases of OpenDaylight and between them the way to install it and run the bundles changes a bit. In this dissertation, although we have developed bundles using all of them, we will only explain how to work with the most updated ones (Helium and Lithium) that are based on Karaf (Chapter 3.1.4).

2. **Create the Maven Project:** The next stage is to make the directory structure of the OSGi bundle. Since OpenDaylight is based on Maven, the best option is to use it also for any new component. So let’s put an example:

```
LearningSwitch
|-- src
    |-- main
       |-- java
       `-- cat
           |-- edu_reina
              |-- LearningSwitch
```

As the name suggests, Java implementations go into the directory LearningSwitch/src/main/java. The directory cat.edu_reina.LearningSwitch is the chosen one for the implementation of my control component.

3. **Creation of the POM.xml** Essential to the Maven build process is a so-called Project Object Model (POM) file called pom.xml that you have to create in the directory `~/LearningSwitch` with the following content (Appendix B.5):

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.reina</groupId>
  <artifactId>LearningSwitch</artifactId>
  <version>1.0.0</version>
  <packaging>bundle</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>3.3.0</version>
        <extensions>true</extensions>
        <configuration>
          <bundleSymbolicName>LearningSwitch</bundleSymbolicName>
          <requires>
            <require>org.opendaylight.openflowjava:*</require>
          </requires>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

First of all a group id is defined (unique id of our organization) and artifact id (name of our component/project) as well as a version number. The packaging element specifies that an OSGi bundle (JAR file with classes, resources, and manifest file) should be built.

During the Maven build process, plugins are invoked. A very important plugin here is the Bundle plugin from the Apache Felix project that creates the OSGi
4.1 An ADSAL Learning Switch: bundle. The import element specifies every package that should be imported by the bundle. The wildcard * imports everything referred to by the bundle content, but not contained in the bundle Apache Felix, which has to be explicitly imported. Moreover, every implementation from the created package has to be exported.

The bundle activator (explained next) is called during the life-cycle of the bundle when it is started or stopped. Basically, it is used to register for services that are used by the bundle component.

The dependency element specifies other packages to which our component 'LearningSwitch' has dependencies. Basically, only the API-driven Service Abstraction Layer (SAL) of OpenDaylight will be necessary. The OpenDaylight project provides its own repository with the readily-compiled components. Thus, Maven will download the JARs from this remote repository.

4. Perform an Eclipse Project: From this POM file, you can now create an Eclipse project by executing:

```
edu@daylight:$ cd ~/LearningSwitch
edu@daylight:$ mvn eclipse:eclipse
```

Note that there is no need to import all the source code of OpenDaylight into Eclipse, just the bundle that is being developed. If any changes are made in the POM, the Eclipse project will have to be rebuilt with the same commands.

5. Import the Project to Eclipse: Afterwards, you can import the project into Eclipse:

- Menu Import / General / Existing projects into workspace
- Select root folder /LearningSwitch
6. Implementing the OSGi Component: We only need two class files to implement the 'LearningSwitch' OSGi component:

- An OSGi Activator registering the component with the OSGi framework.
- A packet handler implementing the control logic and executing actions whenever a packet-in event is received.

7. The Activator: We implement the Activator by creating the file Activator.java in the directory ∼/LearningSwitch/src/main/java/cat/edu_reina/LearningSwitch (Appendix n A.4).

We extend the base class ComponentActivatorAbstractBase from the OpenDaylight controller. In OSGi there are two methods start() and stop() that are called by the OSGi framework when the bundle is started or stopped, respectively. These two methods are overridden in the class ComponentActivatorAbstractBase to manage the life-cycle of an OpenDaylight component. From there, the two methods getImplementations() and configureInstance() are called.

The method getImplementations() returns the classes implementing components of this bundle. A bundle can implement more than one component, for instance, a packet handler for ARP requests and one for IP packets as will be described next.

Method configureInstance() configures the component and, in particular, declares exported service interfaces and the services used. Since an OSGi bundle can implement more than one component, it is important checking which component should be configured with a conditional sentence. (Appendix A.4).

The next step consist in the declaration of the services exported by the LearningSwitch component. As before said, in order to receive packet-in events, the component has to implement the service interface IListenDataPacket. Hence, PacketHandler implements this interface in (Appendix A.4).
Moreover, a name has to be given to the listener (Appendix A.4) using the property salListenerName.

To take an idea of what is happening during the registration, it is interesting to analyze the method setListenDataPacket() of class 'org.opendaylight.controller.sal.implementation.internal.DataPacketService' where it can be seen that the packet handlers are called sequentially. There might be many components that have registered for packet-in events, and is not allowed to force OpenDaylight to call your listener first before another one gets the event. So the order in which listeners are called is basically unspecified. However, it is possible to create dependency lists using the property ‘salListenerDependency’. Furthermore, using the property ‘salListenerFilter’ is viable to set an 'org.opendaylight.controller.sal.match.Match' object for the listener to filter packets according to header fields. Otherwise, all the packets will be received unless another listener consumes it before our handler is called.

Other services are used besides exporting the packet listener implementation. These are declared in (Appendix A.4). To get the object implementing these services to call them, is needed to define two callback functions as part of the component class 'PacketHandler' that will be explained right after. These callbacks are named setDataPacketService() and unsetDataPacketService() and are called with a reference to the service.

8. The Packet Handler: The next part to implement is the packet handler, the class that has been configured through the activator above. This one is the responsible for the reception of packet-in events.

To this end it is implemented the class PacketHandler by creating the following file PacketHandler.java in the directory (Appendix A.5):

`~/LearningSwitch/src/main/java/cat/edu_reina/LearningSwitch`

The (Appendix A.5) contains the detailed explanation of each part of the code.
Consider checking it to deeply understand its behavior. Nevertheless, a schematic representation of the path that the packet-in will do in the PacketHandler is provided here below.

The handler implements the listener interface IListenDataPacket.

```java
public class PacketHandler implements IListenDataPacket
```

This interface declares the function receiveDataPacket(), which is called when a packet-in event from OpenFlow is received.

```java
public PacketResult receiveDataPacket(RawPacket inPkt) { ... }
```

Following, we process the packet received following the diagram:
Figure 4.2: Steps to process a packet in the Packet Handler of an ADSAL bundle
The most important parts of the procedure are introduced below, in order to get a deeper knowledge of the script behavior consider checking the full code in the Appendix A.5.

1. We verify that a packet (inPkt) has been received:
   
   ```java
   if (inPkt == null) {
       return PacketResult.IGNORED;
   }
   ```

2. Then, let’s identify and save the incoming connector(port):
   
   ```java
   NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
   ```

3. The following step is to identify and save the incoming node(switch):
   
   ```java
   Node incoming_node = incoming_connector.getNode();
   ```

4. Next, we extract the packet from inPkt:
   
   ```java
   Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);
   ```

5. We check that it is an Ethernet (L2) packet:
   
   ```java
   if (!(formattedPak instanceof Ethernet)) {
       return PacketResult.IGNORED;
   }
   ```

6. And we extract byte[] srcMAC and dstMAC of Ethernet class:
   
   ```java
   long srcMAC_val = getSourceMACAddress(formattedPak);
   long dstMAC_val = getDestinationMACAddress(formattedPak);
   ```

7. Following, we convert to hex long. We can use BitBufferHelper class to help us.
private long getSourceMACAdress(Packet formattedPak) {
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
    long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    return srcMAC_val;
}

private long getDestinationMACAdress(Packet formattedPak) {
    byte[] dstMAC = ((Ethernet)formattedPak).
    getDestinationMACAddress();
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
    return dstMAC_val;
}

8. Next we check if the packet is ARP → Hub implementation().
   If not → 9.

9. Thus, if it is not an ARP packet, we check if the srcMAC is already saved in our dictionary of MACs.

10. (a) If it is not saved, store it as a key and 'Incoming Connector' as the value in a mapping dictionary.
    
    mac_to_port.put(srcMAC_val, incoming_connector);

    (b) Do not store it again.

11. Verify if we know the destination MAC looking it up in the mapping dictionary.
    
    boolean isDstMACKnown = mac_to_port.containsKey(dstMAC_val);
    if (!isDstMACKnown) {
        ... }

    (a) If it is NOT found -→ Hub implementation().

    } else {
        floodPacket(inPkt);
    }

    (b) If it is found → 12
12. Generate a match for the flow

```
byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
Match match = new Match();
match.setField( new MatchField(MatchType.IN_PORT, incoming_connector) );
match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) );
```

13. Generate an action for the flow

```
dst_connector = mac_to_port.get(dstMAC_val);
List<Action> actions = new ArrayList<Action>();
actions.add(new Output(dst_connector));
```

14. Program a flow rule in switch

```
Flow f = new Flow(match, actions);
```

9. Implementing the OSGi Bundle  Once the component is implemented, it has to be compiled and bundled using Maven. To this goal is only needed to process with the following commands:

```
edu@daylight:$ cd ~/myctrlapp
edu@daylight:$ mvn package
```

Assuming that the POM and the code are correct, this will create the bundle that is the following JAR file:

```
~/LearningSwitch/target/LearningSwitch-0.1.jar
```

10. Run our Karaf distribution:  The next step is to run the Karaf distribution of OpenDaylight that has been configured previously.

```
edu@daylight:$ cd ~/distribution-karaf-0.2.3-Helium-SR3
edu@daylight:$ ./bin/karaf
```
11. Installing the Bundle: In this instant the bundle is ready to be used. Therefore this is the moment to install it in the OSGi framework of OpenDaylight with the following command:

```
bundle:install file/<path for the jar>
```

For example:

```
>opendaylight-user@root> bundle:install file:/home/user/LearningSwitch/target/LearningSwitch-0.1.jar
Bundle ID: 413
>opendaylight-user@root> bundle:list -s
START LEVEL 100 , List Threshold: -1
ID | State | Lvl | Version | Name
----|-------|-----|---------|---------------------
410 | Resolved | 80 | 1.1.3.Helium-SR3 | org.opendaylight.controller.samples.sample-toaster
411 | Resolved | 80 | 1.1.3.Helium-SR3 | org.opendaylight.controller.samples.sample-toaster-consumer
412 | Resolved | 80 | 1.1.3.Helium-SR3 | org.opendaylight.controller.samples.sample-toaster-provider
413 | Resolved | 90 | 0.1.0 | cat.edu.reina.L2Switch
```

As shown, the bundle has been correctly installed. In the list it appears all the bundles installed with different States:

- **Active** → The bundle is in process.
- **Resolved** → The bundle has been stopped or not already started.
- **Installed** → The bundle has just been installed but not used.

If this one or any other bundle are to be deleted, the next commands must be followed:
12. Stop bundles that can interfere  Other bundles that consume packet-in can interfere with the application L2Switch that has been developed. In order to be sure that this does not happen, it is important to stop the bundles that also implement a packet listener, for example, if it has previously installed the L2switch bundle already developed in OpenDaylight.

The command to do it is again:

```bash
opendaylight-user@root>stop <id of the bundle>
```

13. Start the bundle  Observing that Karaf has given the bundle id 413 to the component, it could be used to start it:

```bash
opendaylight-user@root>start 413
```

14. Mininet  For testing, it is possible to use, for example, a linear Mininet topology with one switch and three hosts connected:

```bash
edu@daylight:$ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller remote
```

15. Check the behavior.  An option to check the behavior is to make a ping between hosts, for example:
mininet> h1 ping h2

We can check if the ping reaches correctly its destination. As expected, the first ping has to keep the Address Resolution Protocol (ARP), so it spend more time to reach the destination. Also, this first time, a new flow in the switch is installed. Thus, the next packet will be able to reach the destination without being sent again to the controller. Consequently, all next packets will be sent in tenths of milliseconds, far more quickly than the first one.

16. DLUX OpenDaylight provides a graphical user interface called DLUX. This graphical user interface, not very stable at the moment, is meant to be a powerful tool to interact with the controller and to check the network behavior.

To use it we have to install the following features using the Karaf shell:

```
$ feature:install odl-l2switch-switch
$ feature:install odl-dlux-core
$ feature:install odl-dlux-node
$ feature:install odl-dlux-yangui
$ feature:install odl-dlux-yangvisualizer
```

Once enabled the features, we can access to the interface by opening a browser and entering the login URL: `http://<controllerIP>:8181/index.html`

After that we have to login with a username and a password credential that by default are both admin. At this time we are able to see a graphical picture of the topology of our network (picture XX).

DLUX provides a diverse number of interesting functionalities, e.g. to install manually new flows or send and receive JSON codes to interact with the YANG-based MD-SAL datastore (see Chapter 3.1.3). However, this functionalities have an important quantity of bugs yet, that makes very difficult to work with.
Nevertheless, to check the behavior of our network, we can use three useful functionalities, the Network Topology, the Network Statistics and the Network flows.

![Network Topology](image)

**Figure 4.3: DLUX, The Graphical User Interface of OpenDaylight**

**Viewing Network Topology** The Topology tab displays a graphical representation of the network topology created. We can access it by selecting Topology on the left pane (figureXX). At this point a graphical representation of the network is presented, where the blue boxes represent the switches, the black represents the available hosts, and lines represents how the switches and hosts are connected.

To see the characteristics of each element we can hover the mouse on them.

By using this interface we are able to check the topology of our network.
4.1 An ADSAL Learning Switch:

Viewing Network Statistics  With the 'Nodes module' on the left pane we can view the network statistics and port information of the switches in the network.

Once selected the module, the right pane displays a table that lists all the nodes, node connectors and the statistics.

In our case we only have one switch. But in a more complex topology we could enter a node ID in the Search Nodes tab to search and view the details: port ID, port name, number of ports per switch, MAC Address...

This functionality can help us to check the behavior of our network: we can send different pings by Mininet and check by which ports have been transmitted to.

Viewing Network Flows  If we click now the 'Flows' tab we can view the Flow Table Statistics for the particular node: table ID, packet match, active flows and so on.

Using this option, we can check all the flows that the controller install in the switch: the match (the conditions that a packet has to verify) and the action (what to do if the packet verify this conditions).

17. Wireshark  Wireshark is an open source packet analyzer used to check network behavior. It is an essential tool for network developers. We can run it in every node (s1 in this particular case) by typing:

\[
\text{mininet}> \text{<name of the node>} \text{ wireshark &}
\]

\[
\text{mininet>s1 wireshark &}
\]

Where the symbol & returns us the command line.
With this command we open Wireshark with the possibility to analyze all the interfaces (the ports in our case) of the concerned switch. The program captures the packet in all the interfaces that we specify and we can check its characteristics, for instance the L2 headers (MAC origin and destination addresses...), L3 headers (IP origin and destination addresses...).

It is very useful for troubleshooting: for example, we can observe that a packet does not arrive to the destination after making a ping. Then the correct way to move forward is to find/check the place where the packet has been lost by observing all the interfaces consecutively until we find it. Maybe, the reason why we get an error is that we have not programmed well the match or the action in this specific case.

Take a look to Wireshark site if you are not familiarized with this software. You can find useful documentation and tutorials: https://www.wireshark.org/

4.2 Extension to a Multi Switch Network topology

In order to extent the previously created bundle to a multi-switch topology, not very complex changes are required (Appendix A.6). OpenDaylight, in contrast to other controllers, will not create an instance of a controller for each switch. For these reason, in the bundle that is being created, there is needed to add more options in the mapping dictionary of the already created flows:

**Single Switch Bundle:**

```java
private Map<Long, NodeConnector> mac_to_port =
    new HashMap<Long, NodeConnector>();
```
4.3 Another ADSAL Bundle: Specific for a concrete Topology

Where: Long = source MAC identifier.

NodeConnector = origin port.

Multi Switch Bundle:

```java
private Map<Node, Map<Long, NodeConnector>> mac_to_port_per_switch =
    new HashMap<Node, Map<Long, NodeConnector>>();
```

Where: Node = Switch that is being concerned.

Long = source MAC identifier.

NodeConnector = origin port.

Since the packet-in arrive to the controller for each node, we have to add another Map option in the previous dictionary: the Node. With this new characteristic, each node will have its own dictionary, like in the previous case (one single switch, Chapter 4.1).

Once clarified this, the procedure is very similar. To get a deeper idea check the full code carefully commented in (Appendix A.6).

### 4.3 Another ADSAL Bundle: Specific for a concrete Topology

Aforementioned, one of the reasons why SDN has arisen is to solve each singular case, giving much more power to the particular user rather than the network provider. In order to verify this, let’s solve a hypothetical particular case that could have a specific company:
The company Edu’s Telecom wants to create the following network topology.

Figure 4.4: An specific possible real scenario to develope an OSGi bundle

The way in which the networking department wants to fix the behavior is the following: The Antenna’s Department is authorized to contact directly to the person in charge of the commercial department of Cisco Antennas. However, for obvious reasons, it will not contact with the Optical Cable Corporation agent in the company. If for any case it has to contact him, the communication has to be established across the CEO Department, which wants to know the reason of this act.

Exactly the same but in the opposite way, happens if the Optical Fiber Department wants to contact with anybody from the Commercial Department.
As seen, this topology cannot be done with a L2 learning switch because this situation requires some premises. However we want to use reactive flow programming, which is one of the potentials of ADSAL. This could not be done with the traditional switches, that learned automatically or with previously set rules (proactive flow). This is one of the powers of SDN.

The solution to understand this network consist on taking care of MACs and ports:

The first time a packet arrives to a switch without knowing how to proceed, the packet is sent to the controller. This one, using a switch case structure (see Appendix A.7) classifies the packet considering the following variables: the switch that has received the packet, the source MAC and the destination MAC. Joining these variables, it obtain the nodeConnector (Port of the concerned switch) where the packet has to be sent. Moreover, it installs a new rule in the switch in order to avoid sending each packet with the same header characteristics to the controller. In this way a lot of time is saved going from the tens to the tenths of milliseconds.

4.4 An MDSAL Learning Switch

Notwithstanding the above explained, there is the possibility to develop bundles in the MDSAL architecture. However it is complicated and rarely used by network developers or enterprises. The developers who elaborate MDSAL compatible bundles are normally collaborating with the OpenDaylight Project List and these bundles are open source and for general purposes (e.g L2 Switch, VTN Manager, Topology Manager,..). If you want to collaborate these projects or you want to begin a new one, take a look at the Projects section of the OpenDaylight Wiki\(^1\)

As it usually said, do not reinvent the wheel. It is easier and more useful to

\(^1\)https://wiki.opendaylight.org/view/Project_list\[24\]
use the existing projects than developing our own MDSAL bundle for an specific scenario. Develop the lack of intelligence that we need outside the controller. And finally contact with OpenDaylight by its REST API (Chapter 5 and 6) to connect both parts.

If despite all of this you still want to deploy the MDSAL bundle, it is possible but it is not a subject of this dissertation because of the aforementioned troubles, so I redirect you to the following page where the topic is faced for beginners: [http://sdnhub.org/tutorials/opendaylight/](http://sdnhub.org/tutorials/opendaylight/)

![Figure 4.5: Projects and its dependencies in OpenDaylight Lithium [24]](image-url)
Chapter 5

Outside Programming: The Model-Generated common Rest API

5.1 Introduction

In this section there is detailed the already introduced Model-Generated common Rest API that connects the NB with the SB in MDSAL (Figure 3.4).

Until now, it has been explained how to process a packet when it is captured in the controller. But what happens if we do not have access to program the controller or if the network intelligence is centralized in another program outside the controller? For this kind of situations it has been developed the NB, in order to contact with the outside world.

There are two kinds of NB, the ADSAL one is less developed and it is used to contact from outside applications to the SAL or some bundle.

The other is the MDSAL, one of the essential parts of a SDN controller that
wants to be controlled by outside Network-control-applications. In this case, the outside program implements a control logic interface with the SDN controller and then, it uses the OpenFlow protocol to program the switches according to the instructions that has received through the NB (Chapter 3.5.2, Figure 3.6). This approach will be the one used thereafter because it is the most completed and also because it is prevailing over ADSAL in terms of being used by consumers.

In order to develop all kind of options in each imaginable network, this API should expose all the essential functionalities of OpenFlow to the control application. Unquestionably, the most essential function of SDN is flow programming to define forwarding table entries on the switches. However, it has been seen that there exist two ways to install flows (Chapter 3.4):

- **Reactively**: where OpenDaylight reacts to packet-in events triggered by packets without matching forwarding table entries. It has been widely explained and developed by bundles in Chapter 4.

- **Proactively**: Where the control application, that can be directly the controller or an external application, decides to program a flow on the one hand, without having received any packet-in. It will be further analyzed in Chapter 5.4.

So, if we want a multipurpose API, we should have to implement both approaches. And this is not possible (seen in next section).

### 5.2 Current API

The current API that offers OpenDaylight is a RESTful interface using XML or JSON over HTTP.

RESTful interfaces are extendedly used in many web services and applications,
thing that makes parsing and creating JSON or XML messages and sending or receiving these messages over HTTP is well-supported by many libraries.

As done in the second Chapter, let’s briefly introduce the languages and technologies previously mentioned, that are essential for the NB deployment and usability.

5.2.1 REST

Is short for REpresentational State Transfer. It describes how one system can communicate state with another. In our case would be the state of a flow (its name, type, source port, destination port ...) represented as XML or JSON from a client (outside application) to a server (controller). The generalized idea of state is termed a resource.

5.2.2 JSON

Is short for JavaScript Object Notation. It is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner. In our case will be the representation of a flow that then will be sent by REST.

5.2.3 XML

Extensible Markup Language is, like JSON, another data descriptor. In our particular case could be just a substitute of JSON.
5.2.4 HTTP

Short for HyperText Transfer Protocol. It is a set of standards that allow users of the World Wide Web to exchange information. When accessing any web page entering http:// in front of the address tells the browser to communicate over HTTP. For example, the URL http://<controllerIP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1 will allow us to introduce a new flow interchanging information by HTTP.

Once introduced the components of the REST-API, some limitations have to be exposed. Due to the request/response nature of REST and HTTP, these interfaces are restricted to proactive flow programming. The very essential feature of reacting to packet-in events is missing. The solution for this case should be to program an OSGi bundle, that are the only ones that can get a packet-in implementing IlistenDataPacket interface (Chapter 4.1), and then program yourself the communication with the outside control application.
5.3 Possible solution to solve reactive flow programming

A possible solution for being able to receive packet-in by REST is to implement our OSGi bundle to receive the packet-in and implement a middleware between the controller and the external applications.

This option has been considerate and studied but then discarded for the following reasons:

- Not change standards: we have considered that implementing this solution expose making too many changes inside the OpenDaylight controller while we are trying to get benefit from all its already developed applications.

- SDN is very used to deploy particular networks from specific users in which the topology is already known. Many times in these occasions, we will not need to get the packet-in since with proactive flow programming would be enough to control the network.

- Existence of easier ways to get packet-in indirectly. In the Chapter 6.2 it will be exposed a smart one.

Anyway, an example of this program is available. It is called SDN-MQ (Integrating Message-oriented Middleware and SDN Controller) and it is accessible in the following github address: https://github.com/duerrfk/sdn-mq

It is provided with a slight explanation, but in any case, to get a more precise idea we have elaborated a design of its behavior and its consequent description.
Three bundles/services are implemented by SDN-MQ to date:

- Packet-in service to receive packet-in events including packet filtering based on header fields using JMS selectors.
• Flow programming to define flow table entries on switches.

• Packet forwarding to forward either packets received through packet-in events or new packets created by the application.

The JMS middleware transports messages between the SDN-MQ services and the control applications. It is a system of queues that keep the coordination in order that multiple control applications being able to work with the same packet-in event.

5.4 Installing and reading Flows

Once here, after rejecting the possibility of implementing bundles to get packets-in, it is the moment to explain how to install and read flows from an outside application. Finally, in the next chapter, we will apply this method in a possible particular scenario that a company may have to face. It will also be explained a smart solution to solve the problem of not being able to get packets-in through REST-API.

The common REST-API changes completely between the old and the new (Karaf distributed) releases of OpenDaylight. In this project both ways have been deeply studied and developed, however the Karaf one should be explained more accurately for obviously reasons: it is the most updated release.
Figure 5.2: Steps to install and read flows from outside by REST through Open-Daylight
5.4 Installing and reading Flows

5.4.1 Settings

1, 2. Run the controller: Run the controller as explained in (Chapter 4.1):

Get the OpenDaylight official helium distribution. Unzip the Karaf distribution and go to bin directory of your distribution and run following command to start Karaf. It will start the Karaf console, which may not have any feature installed by default.

$ ./karaf

In case of an Hydrogen distribution: ./distributions/target/mdsal/run.sh

3. Important features: In Karaf based distributions, some features have to be installed to enable the REST-API and the DLUX that we will use later to see the flows and the topology. As well as this, it is also useful to enable the API-DOC of the controller, where all the REST interfaces can be consulted.

Thus, install basic MD-SAL controller features on the Karaf console. It is recommended installing the following features before starting the DLUX feature. L2Switch feature internally enables MD-SAL data broker and Openflow plugin service. L2Switch also makes sure that in topology UI, hosts are also visible along with switches. We need the mdsal-apidocs feature for YangUI in DLUX.

$ feature:install odl-restconf

$ feature:install odl-l2switch-switch

$ feature:install odl-mdsal-apidocs

4. Helpful features: Install the AD-SAL features on the Karaf console. They are not essential but they will be useful when we check the behavior of the network in the DLUX API.
5. Installing DLUX  Then, install the DLUX feature to check the topology and the flows

$ feature:install odl-dlux-core

6. Mininet  Run Mininet with the topology that we want to control, fixing the remote controller in order to establish the connexion with the controller.

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

For more details about how to configure and run Mininet with support for OF 1.3, visit the next link in the wiki of OpenDaylight:

https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin::Test_Environment

5.4.2 Postman REST-Client

7. Run Postman REST-Client  Postman is a REST Client that runs as an application inside the Chrome browser. It is very useful for interfacing with REST APIs such as the once of OpenDaylight. In the Chrome browser navigate to https://chrome.google.com/webstore then search for Postman and it will show as offered by www.getpostman.com
8. Flow Strategy  OpenDaylight Karaf distributions have the REST common API divided in two halves. The 'restconf/config' and the 'restconf/operational'.

1. We will publish the flow in the 'config' once. In this API it is possible to add non valid values for each key (the keys must be valid). Thus, we have to be careful, such no value validation is done here.
   
   This information will be committed into the dataStore.

2. If suitable provider for the node where we want to install the flow is provided (for example the switch is available in Mininet), the Flow Programmer Service will check if all the pairs key-value that we are trying to install in this node are valid. In the case all values were correct, the flow is published to the 'restconf/operational' API, also saved in the Flow Service Model Data.
3. Once the flow is in the ‘restconf/operational’ API, The provider (plugin in this case) transforms MD-SAL modeled flow into OF-API modeled flow.

4. OF-API modeled flow is then flushed into OF Library

5. OF Library encodes flow into particular version of wire protocol and sends it to particular switch

6. Check on Mininet side if flow is set

9. **Check the name of the nodes**  OpenDaylight give a random name to the nodes (switches) of our network following the next syntax (openflow:X). If it is true that in the lasts releases the nodes are in growing order beginning from 1, as it should be expected, in older releases this does not happen. To check the names of the switches, just point the browser to the next URL in the ‘operational’ API:

   http://<controller_IP>:8181/restconf/operational/opendaylight-inventory:nodes/

   where we can see a long list of information about the nodes:
In our case, this node is called openflow:1.

**10. PUT the flow**  Use Postman to set the following values.

Set headers:

Content-Type: application/xml

Accept: application/xml

Authentication

URL: http://<controllerIP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1

PUT

Use Body:
As has been said previously in the ADSAL approach, to install a flow we have to establish a match (conditions that the packet arriving to the switch have to achieve) in order to set an action to do with this packet. In the present case, all the IP packets (ethernet-type = 2048) with the IP source 10.0.0.1/32 and the IP destination 10.0.0.2/32, will be sent by the Port 2 (output-node-connector) of the

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <priority>10</priority>
    <flow-name>S1-S2</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-source>10.0.0.1/32</ipv4-source>
        <ipv4-destination>10.0.0.2/32</ipv4-destination>
    </match>
    <id>2</id>
    <table_id>0</table_id>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>2</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```
switch.

It has to be stressed that the mask of the IP is mandatory to install the flows, in the contrary case, the flow will be published in the 'config' API, but not translated to the 'operational' API, so the switch will not see any new flow. In this REST-API, we dispose of multiple tables to install the flows. Each flow will be characterized by the table in which it is installed and its ID, that can be an alphanumerical value. However, if we want to try a different flow id or a different table, we have to be sure that the URL and the body stay in sync. For example, if we wanted to try: table 2, flow S1_ARP; we would change the URL to:

\[
\text{http://<controllerIP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/2/flow/S1_ARP}
\]

but we would also need to update the \(<\text{id}\)S1_ARP</id> and <table_id>2</table_id> in the body of the XML.

11. Check for the flow on the 'operational' API If the flow has correct values, it has to be correctly installed in the 'operational' API. To check it, is as easy as point our browser to the same link where we have put the flow, but changing 'config' for operational:

\[
\text{http://<controllerIP>:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/2/flow/1}
\]

It can also be checked by a Postman call:

\[
\text{URL http://<controllerIP>:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/}
\]

GET

You no longer need to set Accept header.
The return response will be the same that can be observed accessing by the browser.

12. **Check for the flow on the switch**  As soon as the flow has been inserted on the 'operational' API, the switch has seen it and installed in its own forwarding table. To check it exist the OpenFlow command that we can insert by the shell:

```
mininet@mininet-vm:$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=7.325s, table=0, n_packets=0, n_bytes=0,
idle_timeout=300, hard_timeout=600, send_flow_removed priority=2, ip,
nw_dst=10.0.10.0/24 actions=dec_ttl
```

If we want to see the above information from the Mininet prompt, we have to use `sh` instead of `sudo` i.e. `sh ovs-ofctl -O OpenFlow13 dump-flows s1`.

13. **Check the behavior**  To check the behavior it is possible to make a ping between hosts, for example:

```
mininet> h1 ping h2
```

14. **DLUX**  OpenDaylight provides a graphical user interface called DLUX. This graphical user interface is meant to be a powerful tool to interact with the controller and to check the network behavior. Thus, DLUX provides a diverse number of interesting functionalities, e.g. to install manually new flows or send and receive JSON codes to interact with the YANG-based MD-SAL datastore (see Chapter 3.5.2). However, this functionalities have an important quantity of bugs yet, that makes very difficult to work with. Nevertheless, to check the behavior of our network, we can use three useful functionalities that it provides: the Network Topology, the Network Statistics and the Network flows. For more information
about how to install and use the interface, as well as all its functionalities, take a look to Chapter 4.1.16.

15. Wireshark  Wireshark is an open source packet analyzer used to check network behavior. It is an essential tool for network developers:

The program captures the packet in all the interfaces that we specify and we can check its characteristics like the L2 headers (MAC origin and destination addresses...), L3 headers (IP origin and destination addresses...)... Thus, it is very useful for troubleshooting.

For more information about how to install and use the interface, as well as all its functionalities, take a look to Chapter 4.1.17.

16. Installing all kind of flows, the API-DOC  Once the procedure to install a flow is known, to fully understand how to install other kinds of flows, it is useful to visit the API-DOC of the controller (Figure[4.5]). It is a Documentation of all the REST calls that OpenDaylight controller admits.

As installed before in the step 4, we only have to point our browser to the following direction:

http://<controller_IP>:8181/apidoc/explorer/

To install new flows we will use the 'inventory' section.
17. Create collections in Postman  Postman offers the possibility to create collections where we can save all the instructions mentioned to PUT, GET and DEL flows. It is as easy as click the ‘Collection’ button in the left-top of the screen and follow the instructions:

- To give a name for the collection.
- To write a description about the collection.
- To add all the instructions that we want, for example the PUT that we have used in Chapter 5.4.10.
5.4 Installing and reading Flows

We can see some screen shots of postman in the Appendix B.2.

5.4.3 Java Code

18. Java code to control and install flows  Sometimes, it is very useful to create a code in Java to generate the same flows and send them exactly as we were doing by Postman.

With Postman we can only sent proactive flows, but what happened if we want to develop an intelligence adaptive network, where the networks modify its behavior depending on the traffic? For this case, that will be widely exposed in the next chapter (Chapter 6), it will be indispensable to send the RESTful calls by a Java (or another language) application.

Creating the JSON flows: The RestInterfaceJSONCreator

It is used to create the body of the flow. However, instead of doing it by xml as we were doing with Postman, we will use JSON in order to get help from its libraries for Java. In the API-DOC is explained the structure that a flow has to follow in JSON. It is also useful to firstly insert the flows with Postman and then call the flow (GET) receiving the answer in JSON. Finally reproduce this JSON with Java instructions. The classes that we will use to create the JSON are the following:

```java
JSONObject outputAction1 = new JSONObject();
JSONArray action = new JSONArray();
```

For example, following the API-DOC, when we see that some instructions are comprised between the square brackets ‘[ ]’, we know that everything inside is a JSONArray. In the other hand, when it is comprised between braces ‘{ }’, it is a
JSONObject. Let's see a piece of the code for a better comprehension:

<table>
<thead>
<tr>
<th>Java Code</th>
<th>JSON</th>
</tr>
</thead>
</table>

```java
JSONObject outputAction1 = new JSONObject();
outputAction1.put("output-node-connector", "3");

JSONObject outputAction2 = new JSONObject();
outputAction2.put("output-node-connector", "5");

JSONObject outputAction3 = new JSONObject();
outputAction3.put("output-node-connector", "5");

JSONObject insideAction1 = new JSONObject();
insideAction1.put("order", 0);
insideAction1.put("output-action", outputAction1);

JSONObject insideAction2 = new JSONObject();
insideAction2.put("order", 1);
insideAction2.put("output-action", outputAction2);

JSONObject insideAction3 = new JSONObject();
insideAction3.put("order", 2);
insideAction3.put("output-action", outputAction3);

JSONArray action = new JSONArray();
action.put(insideAction1);
action.put(insideAction2);
action.put(insideAction3);
```

```json
{
    "action": [
        {
            "order": 0,
            "output-action": {
                "output-node-connector": "3"
            }
        },
        {
            "order": 1,
            "output-action": {
                "output-node-connector": "5"
            }
        },
        {
            "order": 2,
            "output-action": {
                "output-node-connector": "5"
            }
        }
    ]
}
```
Once all the JSON flow is deployed, we have to set the URL where the flow has to be installed. Obviously, it is the same URL that we have used to install the flow by Postman:


The last step is to instantiate the class RestInterfaceSender, that we will create below, and send it the JSON flow and the String baseURL.

Sending the JSON flows: The RestInterfaceSender

It is used to establish the connexion with the OpenDaylight REST-API in order to send the JSON flow that has been created with the RestInterfaceJSONCreator. We implement a method that will receive the String URL and the JSON flow parameters:

```java
public static boolean installFlow(String baseURL, JSONObject flow) {

    The first thing to do is to translate the URL where to PUT the flow creating an URL Java Object of the java.net.URL library.

    Then, we have to set what in Postman where the Method, the authentication and the headers. To do that we will need the org.apache.commons.codec and the org.codehaus.jettison.json library. Once the connection with all the required values is established, we send the JSON that we have created with the RestInterfaceJSONCreator and we write it into the "config" API.

    We ask about the result of the transfer and a REST code will be received. This code let us know the result of the REST PUT. If everything has gone OK, we will receive a 200 Status code.
<table>
<thead>
<tr>
<th>Status code</th>
<th>Error message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>None</td>
<td>This status code is returned on successful GET operation.</td>
</tr>
<tr>
<td>201 (Created)</td>
<td>None</td>
<td>This status code is returned on successful POST operation.</td>
</tr>
<tr>
<td>204 (No Content)</td>
<td>None</td>
<td>This status code is returned on successful PUT or DELETE operation.</td>
</tr>
<tr>
<td>400 (Bad Request)</td>
<td>Invalid Format</td>
<td>This status code is returned if the format of request body is not valid.</td>
</tr>
<tr>
<td>401 (Unauthorized)</td>
<td>Unauthorized</td>
<td>This status code is returned if the user name or password is not valid.</td>
</tr>
<tr>
<td>403 (Forbidden)</td>
<td>Forbidden</td>
<td>This status code is returned if the user has no authority to do the operations.</td>
</tr>
<tr>
<td>404 (Not Found)</td>
<td>Not Found</td>
<td>This status code is returned if a part of URI other than arguments is incorrect.</td>
</tr>
<tr>
<td>405 (Method Not Allowed)</td>
<td>Method Not Allowed</td>
<td>This status code is returned if method is not supported for specified URI.</td>
</tr>
<tr>
<td>403 (Forbidden)</td>
<td>Forbidden</td>
<td>This status code is returned if the user has no authority to do the operations.</td>
</tr>
<tr>
<td>406 (Not Acceptable)</td>
<td>Not Acceptable</td>
<td>This status code is returned when Accept header field is specified as other than ”application/json” or ”application/xml”.</td>
</tr>
<tr>
<td>409 (Conflict)</td>
<td>Duplicate Entity Exists (\text{name}_i;\text{value}_i)</td>
<td>This status code is returned when resource already exists. Note: \text{name}_i is parameter name and \text{value}_i is the first 1024 characters of invalid parameter value.</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>415</td>
<td>(Unsupported Media Type)</td>
<td>This status code is returned when Content-Type header field is specified as other than &quot;application/json&quot; or &quot;application/xml&quot;.</td>
</tr>
<tr>
<td>500</td>
<td>(Internal Server Error)</td>
<td>This status code is returned when an internal error during processing has occurred.</td>
</tr>
<tr>
<td>503</td>
<td>(Service Unavailable)</td>
<td>This status code is returned when the web server is busy or not available, usually due to maintenance or server overloading.</td>
</tr>
</tbody>
</table>
Chapter 6

The External Orchestrator: an Intelligent outside application to manage networks

Putting the intelligence in an outside application is incredibly useful in a wide range of occasions. Let’s refresh what was said in the previous chapter:

- Sometimes we cannot access to the controller for physical reasons, we do not have rights...
- In some occasions the intelligence of the network is already centralized in another program outside the controller for portability reasons, because is written in another language...
- We do not want to change a lot the default functionalities of the controller because of the wishes of standard applications.

To illustrate these reasons we will implement an outside application that, with the help of another external program, a DPI, will control a specific network.
A DPI (Deep Packet Inspection) is a program used to classify the packets in order to filter them a posteriori. In our case, the DPI inspects each packet that captures and returns a chain composed by the following parameters:

\begin{Verbatim}
%@IPsrc;<srcPort>;%@IPdst;<dstPort;L4Protocol>
<idProtocolNUM;App>
\end{Verbatim}
6.1 The scenario

Let's explain this hypothetical network structure that has been implemented by Edu’s Telecom.

- **S1** is the switch of the company that has direct access to internet.
- The packets of internet can come from two different networks, which in this example will be approximated by hosts **H1** and **H2**. In a final section we will make the extrapolation to two different networks.

**Figure 6.1:** Hypothetical scenario controlled by an External Orchestrator
• In S3 there is a restrictive Firewall where the dangerous packets will be analyzed.

• H3 is the website of the company.

• H4 is the Sales Department where they do all the economical transactions.

• Edu’s Telecom do not want to have any security problem, especially in the Sales Department, since it manages billions of dollars. For this reason it contacts with another enterprise specialized in security that have a DPI which will analyze all the packets arriving to S3 and will reply with its characteristics to Edu’s telecom.

• To decide what to do with the packets, Edu’s telecom has developed an External Orchestrator. This application, that knows the characteristic of each packet sent thanks of the NDP, takes a decision about how to route all the packets arriving to the network. When this decision has been taken, it creates flows and it sends them to the controller by the REST interface.

### 6.2 Reactive and Proactive capability

REST calls cannot get packet-in events by themselves, they have to implement an API for this reason (Chapter 5.3). However, we can think a smart way to reproduce this reactive flow programming behavior, implementing a pseudo-packet-in handler. This is the case of this External Orchestrator.

1. The External Orchestrator send the ARP flows (L2) in order that all the components could know themselves.

2. The External Orchestrator sends L3 flows. These flows tells the net that each packet arriving to S1 has to be sent to the Security enterprise, and also, has to be duplicated and sent to its first target (@IPdst). This behavior is
chosen because Edu’s Telecom consider that with only a few packet/s the security cannot be damage and it is not wanted to stop the connection while the packet is being analyzed.

3. The DPI captures the packet and analyze it. Next, it creates a TCP Socket client that connects with the TCP Socket Server that has been implemented in the External Orchestrator. Through this connection it sends all the information of the packet.

4. The External Orchestrator gets the information by a Server Socket, which is our particular Packet-in Handler. Then use it to establish new routing rules that it sends through the REST-API to the controller.

5. The next time that a packet with the same characteristics arrive to S1, it will be routed following the rules that the External Orchestrator has decided for it.

If we observe the network performance, can be seen that the External Orchestrator receive packets, in this case not packet-in events, but its information (even more useful for our goal). For this reason, we can state that the External Orchestrator implements both, proactive and reactive flow programming.

6.3 Three kinds of flows

The External Orchestrator implements three kinds of flows to the plain control of the network. One L2 flow to control the ARP and two L3 flows:

6.3.1 L2 flows (ARP)

Each Network needs an ARP protocol in order that all the components of the network know each other. That is, we have to install a protocol for mapping the
Internet Protocol address (IP addresses) to the physical machine addresses that are recognized in the network.

In our case we program it proactively, just when the application is initialized we will send this L2 flows through REST to the controller.

Another possibility to work out the address resolution is to install the L2 Switch bundle in OpenDaylight and it will deal with ARP for its own. In this case the ARP will be reactively resolved and we will have to pay attention to install the next flows with more priority that the once L2 Switch bundle give to ARP once.

$ feature:install odl-l2switch-switch

6.3.2 L3 Initiation flows

When a new packet with different characteristics from previous ones arrive to S1, it has to be duplicated and sent to its destination but also to the DPI to be analyzed. To reach this goal a flow has to be installed in the controller. Thus we will do it proactively after the introduction of the ARP flows. We will put a higher priority to this packets that L2 flows.

6.3.3 L3 definitive flows

After being analyzed by the DPI and sent to the External Orchestrator to make a decision, the next packet with the same characteristics will be flown directly through the path that ONCA had decided, its definitive path (if it is considered dangerous by S3 and if not directly to S2).

These ones will be the third kind of flows installed in OpenDaylight. They will have the highest priority to substitute the initiation flows. Unlike the previous ones, in this case it is reactive flow programming using the REST interface.
6.4 The Structure

Once knowing the behavior of the External Orchestrator, it is the moment to find out its structure and composition. The following diagram is a representation of its components. Each blue box is a Java Class and the arrows and numbers depict the temporal execution.

![Diagram of External Orchestrator structure]

Figure 6.2: Our External Orchestrator structure
6.4.1 GUI User Interface\(^1\) (1\(^2\))

This Class provide a graphical interface in order to make the application more user-friendly. It is used for the set-up, to run the orchestrator and also for the feedback (it procures a screen with the echo of its running services).

![GUI User Interface of the External Orchestrator screenshot](image)

Figure 6.3: GUI User Interface of the External Orchestrator screenshot

6.4.2 Settings\(^3\) (2)

In this class the aforementioned settings that the user prearrange in the GUI interface are set up. They are used for the correct behavior of the orchestrator.

---

\(^1\)The Appendix A.9 contains the full scrip of the GUI User Interface properly commented

\(^2\)From now on, the numbers inside the parentesis are corresponding action in the Figure 6.2

\(^3\)The Appendix A.10 contains the full Settings scrip properly commented
6.4 The Structure

6.4.3 Main External Orchestrator\(^4\) (3)

MainExternalOrchestrator.java is, bridging the gaps, the SAL of our application. It is used to call all the functions and control the behavior of the app. As we can see in the picture (GraphicXX) it is the heart of program.

6.4.4 Flow Initializer\(^5\) (4)

This Class is used to install all the ARP flows and the IP flows for the setup of the network.

**ARP L2 flows**

First of all, the proactive ARPs are set. As we can see the topology which we are dealing with is a ring topology. This kind of topologies can cause a lot of troubles in internet because of the ARP overflow. To avoid this we will implement a circular unidirectional ARP clockwise traffic. This means that:

- Every ARP packet that comes from S2 to S1, will be sent to S3.
- Every ARP packet that comes from S1 to S3, will be sent to S2.
- Every ARP packet that comes from S3 to S2, will be sent to S1.

To fully comprehend how to install the circular traffic, let’s show a table:

\(^4\)The Appendix A.11 contains the full script of the Main External Orchestrator properly commented

\(^5\)The Appendix A.12 contains the full Flow Initializer script properly commented
To install the flow we call the `flowInstallDPI` method of `RestInterfaceJSONCreator.java` (3), setting the following parameters:

```java
newFlowIP.flowInstallDPI(String actionToDo1, String actionToDo2,
                          String actionToDo3, int priority, String name, int etherType,
                          String inPort, String nwSrc, String nwDst, String nodeToInstall)
```

For example, the values for the first flow in the tableXX (Node: openflow:1, Source Port 1) are the following:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionToDo1</td>
<td>2</td>
<td>Port to flow the packet</td>
</tr>
<tr>
<td>actionToDo2</td>
<td>4</td>
<td>Port to flow the packet</td>
</tr>
</tbody>
</table>
### The Structure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actionToDo3</code></td>
<td>-1</td>
<td>Port to flow the packet. We sent a -1 because we do not want to use it, we only need two output ports.</td>
</tr>
<tr>
<td><code>priority</code></td>
<td>501</td>
<td>The priority of this flow has to be higher than the one that send the packet-in to the controller, but lower that the IPs flows</td>
</tr>
<tr>
<td><code>name</code></td>
<td>S1:P1:arpFlowInsert</td>
<td>Name and the ID of the flow. In fact, the name and the IP can be different, but it is easier to manage the flows if we set them equally.</td>
</tr>
<tr>
<td><code>etherType</code></td>
<td>-1</td>
<td>Type of the packet. We do not set it because with ARPs flows we do not need it. Thus, in this moment we are setting rules for all the packets only taking care of its Port origin, but as the L3 flows that we will put after will have more priority, they will filter the packets taking care also about its Type.</td>
</tr>
<tr>
<td><code>inPort</code></td>
<td>1</td>
<td>Port of origin of the packet.</td>
</tr>
<tr>
<td><code>nwSrc</code></td>
<td>-1</td>
<td>Source IP. It is not used in the ARP flows (-1).</td>
</tr>
<tr>
<td><code>nwDst</code></td>
<td>-1</td>
<td>Destination IP. It is not used in the ARP flows (-1).</td>
</tr>
<tr>
<td><code>nodeToInstall</code></td>
<td>openflow:1</td>
<td>Node / Switch to install the flow.</td>
</tr>
</tbody>
</table>

It has to be said that a lot of these values are set in Settings.java and then called from there, not written directly in the function call. It is made with the purpose of save time if we have to deal with a network with different IPs, ports, nodes... In this case we will only need to change these parameters once, in the Settings Class.
L3 Initiation Flows

We do exactly the same has been done with the L2 flows, but changing some parameters. These flows are used to provide an initial behavior to the network (explained widely in Chapter 6.3.2). To reach it, we have to configure it with the following values.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionToDo1</td>
<td>2</td>
<td>Port to flow the packet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node (Switch)</th>
<th>IP source</th>
<th>IP destination</th>
<th>Node Connector (Port)</th>
</tr>
</thead>
<tbody>
<tr>
<td>openflow:1</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>openflow:1:2</td>
</tr>
<tr>
<td></td>
<td>10.0.0.3</td>
<td>10.0.0.4</td>
<td>openflow:1:3</td>
</tr>
<tr>
<td></td>
<td>10.0.0.4</td>
<td></td>
<td>openflow:1:3</td>
</tr>
<tr>
<td>openflow:2</td>
<td>10.0.0.1</td>
<td>10.0.0.1</td>
<td>openflow:2:1</td>
</tr>
<tr>
<td></td>
<td>10.0.0.2</td>
<td>10.0.0.2</td>
<td>openflow:2:2</td>
</tr>
<tr>
<td></td>
<td>10.0.0.3</td>
<td>10.0.0.3</td>
<td>openflow:2:1</td>
</tr>
<tr>
<td></td>
<td>10.0.0.4</td>
<td>10.0.0.4</td>
<td>openflow:2:2</td>
</tr>
<tr>
<td>openflow:2</td>
<td>10.0.0.1</td>
<td>10.0.0.1</td>
<td>openflow:2:3</td>
</tr>
<tr>
<td></td>
<td>10.0.0.2</td>
<td>10.0.0.2</td>
<td>openflow:2:3</td>
</tr>
<tr>
<td></td>
<td>10.0.0.3</td>
<td>10.0.0.3</td>
<td>openflow:2:3</td>
</tr>
<tr>
<td>openflow:2</td>
<td>10.0.0.1</td>
<td>10.0.0.1</td>
<td>openflow:2:1</td>
</tr>
<tr>
<td></td>
<td>10.0.0.2</td>
<td>10.0.0.2</td>
<td>openflow:2:1</td>
</tr>
<tr>
<td></td>
<td>10.0.0.3</td>
<td>10.0.0.3</td>
<td>openflow:2:1</td>
</tr>
</tbody>
</table>
### 6.4 The Structure

<table>
<thead>
<tr>
<th>actionToDo2</th>
<th>-1</th>
<th>Port to flow the packet. We sent a -1 because we do not want to use it, we only need one output port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionToDo3</td>
<td>-1</td>
<td>Port to flow the packet. We sent a -1 because we do not want to use it, we only need one output port.</td>
</tr>
<tr>
<td>priority</td>
<td>601</td>
<td>Priority of this flow, has to be higher that L2 flows (ARP), but lower that the L3 definitive flows.</td>
</tr>
<tr>
<td>name</td>
<td>S1:H1-H2: InitialFlowInsert</td>
<td>Name and the ID of the flow. In fact, the name and the IP can be different, but it is easier to manage the flows if we set them equally.</td>
</tr>
<tr>
<td>etherType</td>
<td>2048</td>
<td>Type of the packet. In this case, IPv4 packet.</td>
</tr>
<tr>
<td>inPort</td>
<td>-1</td>
<td>Port of origin of the packet. It is not used in these flows (-1) because we are only taking care about the L3 source and destination to make the match.</td>
</tr>
<tr>
<td>nwSrc</td>
<td>10.0.0.1</td>
<td>Source IP of the packet.</td>
</tr>
<tr>
<td>nwDst</td>
<td>10.0.0.2</td>
<td>Destination IP of the packet.</td>
</tr>
<tr>
<td>nodeToInstall</td>
<td>openflow:1</td>
<td>Node / Switch to install the flow.</td>
</tr>
</tbody>
</table>

#### 6.4.5 Rest Interface JSON Creator\(^6\) (5)

This class is used to create the body of the flow with JSON and the URL of the REST-API where to install the flow. It has been widely explained in the previous section (Chapter 5.4.3). Once it has created the JSON and the URL, it calls RestInterfaceSender.java (4,17).

---

\(^6\)The Appendix A.13 contains the full Rest Interface JSON Creator script properly commented
6.4.6 Rest Interface Sender\(^7\) (6)

This class is used to establish the connexion with the OpenDaylight REST-API in order to send the JSON flow that has been created with the RestInterfaceJSONCreator.java(3). It has been widely explained in the previous section (Chapter 5.4.3).

Once this function has been called to install all the L2 flows (ARP) and all the L3 initializing flows, the program comes back the main function (MainExternalOrchestrator.java (7)) that will instantiate the next class, in this case the socket server (StockSocketQuoteServer.java).

6.4.7 Stock Socket Quote Server\(^8\)(8)

This Class is a Server Socket. A Socket is an endpoint in a communication flow between two programs running over a network, in this case the DPI and the External Orchestrator. As it is the server part, it will be listening.

Once the L2 flows and the L3 initializing flows are installed, it is the moment to install the L3 definitive flows. For this purpose we need the information portioned by the DPI. To receive this information we have to open a Server Socket in a specific port of our localhost, which will be listening what the DPI will send by a Client Socket.

When we receive the information from the DPI, we send it to the Main class to go on (9).

\(^7\)The Appendix A.14 contains the full Rest Interface Sender scrip properly commented
\(^8\)The Appendix A.15 contains the full Stock Socket Quote Server scrip properly commented
6.4 The Structure

6.4.8 Chain Reader\(^9\)(10)

Once we have received the chain proportioned by the DPI, is the moment to analyze it. We receive a chain like this:

\[
\langle @\text{IPsrc} : \langle \text{srcPort} \rangle : \langle @\text{IPdst} : \langle \text{dstPort} \rangle : \langle \text{L4Protocol} \rangle : \langle \text{idProtocolNUM} \rangle : \langle \text{App} \rangle \rangle
\]

Or with even more parameters if the packet was an ICMPV6 or MDNS packet.

The goal of ChainReader.java is to analyze this chain and return the ports per each switch where the packets have to be routed, not only for the request path, but also for the reply. The route is set taking care about the @IP source, the @IP destination and the application that has generated it (Mozilla, Explorer, Spotify, Facebook,..). Thus, ChainReader.java will decouple this parameters from the chain sent by the DPI and will call other functions from the FullMatrix.java Class to reach the output ports for each node.

6.4.9 Full Matrix\(^{10}\)(11)

It is a Class provided with a matrix that says to which output port has to be sent each packet. Its composition is the following:

\[
\text{[Number of the Switch]}[\text{@IPs}][\text{@IPd}][\text{Type of protocol}] = \text{Output Port}
\]

It is in this moment when the decision of if a packet is or not dangerous is made. For example if arrive the following packet we can consider that is safe and sent it directly to its destination, the website of Edu’s Telecom, flowing it to the S2 by the port 3.

\(^9\)The Appendix A.16 contains the full Chain Reader scrip properly commented
\(^{10}\)The Appendix A.17 contains the full Full Matrix scrip properly commented
Otherwise, if the packet that arrives is the following, it can be considered dangerous and has to be sent to S3 to be further analyzed by the Firewall.

In this case, the packet has to be sent to the port number 4 of the first switch.

This procedure is reproduced exactly for the other two Switches, and also for the three switches again for stipulate the comeback path (the reply).

Once the choice of the port has been decided for all the switches for the request and for the reply, a matrix with all these values is returned, through ChainReader.java (12), to the MainExternalOrchestrator.java (13).

### 6.4.10 Flow Repetition Control (13)

When we install the definitive L3 flows, we can proceed in two ways:

- We can duplicate them and send a copy to the DPI as we have done before with the L3 initializing flows. The advantage of this approach is that we will always receive the information of the packets arriving to the Edu’s Telecom network (S3). Thus, we will be able to implement other functionalities and also change its route afterwards. The disadvantage is that a flow repetition control will be needed.

- We can install them without duplication (without sending a copy to the DPI). This implies not having to install a flow repetition control since the packets will never come back to the External Orchestrator. On the other hand, we
will never again have the information of the packets in the intelligence center, so no more transactions will be able to be made subsequently.

The first method is chosen because of its capability to be modified. For this reason a flow repetition control has been implemented:

When the information of a packet arrives to the External Orchestrator, this flow repetition control is activated. This function controls that the present flow had not been installed previously to not overwrite it, which would mean a waste of time and flow statistics.

To reach this goal, we can adopt two different options:

- Make a dictionary of each flow installed and save it in the External Orchestrator. Then check that each new arriving flow is not in the dictionary.
- Ask directly to OpenDaylight if this flow has been installed before through a REST call. In this case we will use the database of the controller (the Flow Service Data Model) which save all the flows installed (Chapter 3.5.2).

Both methods have been implemented and the user have the option to select which one wants to use by selecting it in the GUI User Interface (1). Knowing this, the External Orchestrator calls the Flow Model (14) to continue with the flow repetition control.

### 6.4.11 Flow Model\(^{11}\) (14)

This Class is a Java Model to introduce the most characteristics parameters to distinguish flows. It is used in various Classes of the External Orchestrator, but probably the most important time is when it is used to make the repetition control.

\(^{11}\)The Appendix A.18 contains the full Flow Model scrip properly commented
control. We instantiate this Class to make a Dictionary of the flows that we have installed. There are two ways to fill this dictionary according with both methods explained above:

- In the case that we were using the first method, we fill this dictionary with an entry each time that we install a new flow.

- In the second case, we use RestInterfaceReciever.java to fill the dictionary.

The model is composed by the following Strings, which characterize an L3 flow, with its according getters and setters (see Appendix A.18):

```java
private String name;
private String nwSrc;
private String nwDst;
private String installInHw;
private String priority;
private String etherType;
```

### 6.4.12 Rest Interface Receiver\(^\text{12}\) (15)

We can activate this Class in Settings.java through the GUI User Interface. If we activate it, the flow repetition control is made by asking the REST-API. If not, by a flow already installed dictionary that we have to develop (see picture 6.3).

This Class is the brother of Rest Interface Sender(6) (Chapter 5.4.3). It is used to establish the connection with OpenDaylight REST-API in order to ask the flows already installed. The first thing is create the URL Java Object where to ask (GET) the flow. The java.net.URL library will help us in this purpose.

\(^\text{12}\)The Appendix A.19 contains the full Rest Interface Receiver scrip properly commented
As we want to obtain all the flows installed in the controller, we ask the ‘rest-conf/operational’ API. Specifically:


[1]We set the table where we install the flows in Settings.java through the GUI User Interface (Image above XX)

Each flow is installed in every switch. For this reason we GET the flows in only one switch to avoid repetitions. The openflow:1 node has been chosen, but could be chosen any other node/switch.

Then, we have to set Request Method (GET), the authentication and the Request Property ("Accept", "application/json"). To do it we will need the org.apache.commons.codec and the org.codehaus.jettison.json library. Next, we establish the connection and we initialize a buffer where all the information will be thrown. We read line by line the buffer in order to write a JSON Object with all the information of the URL consulted.

Finally this JSON Object is sent to Rest Interface JSON Analyzer.

### 6.4.13 Rest Interface JSON Analyzer

This Class receive the JSON Object with a lot of information about the nodes, the statistics, the configuration, the capabilities, the manufacturer, the tables, the flows...

Our goal is to decrypt this information in order to get the flows installed in the controller and save them in the dictionary of the already installed flows.

\[13\]The Appendix A.20 contains the full Rest Interface JSON Analyzer script properly commented
The JSON Object that is received, is composed by many JSON Arrays and JSON Objects. We will have to go across all this structures to arrive until our goal: the id, @IP source and @IP destination of each flow.

Once we get this information, we saved in a dictionary of Flow Models (14).

```java
FlowModel flowInArray[] = new FlowModel[55];
flowInArray[i] = new FlowModel();
```

An example to get the following flow and save it in the Flow Model dictionary is written below:

<table>
<thead>
<tr>
<th>Id</th>
<th>ICMP:from:10.0.0.1:dest:10.0.0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>@IP source:</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>@IP destination</td>
<td>10.0.0.3</td>
</tr>
</tbody>
</table>

Code to extract the information:

```java
FlowModel flowInArray[] = new FlowModel[55];
JSONArray array = myFlows.getJSONArray("flow-node-inventory:table");
JSONObject array2 = array.getJSONObject(0);
JSONArray array_flow = array2.getJSONArray("flow");
JSONObject flow = new JSONObject();
flowInArray[1] = new FlowModel();
flow = array_flow.getJSONObject(i);
flowInArray[1].setName(flow.getString("id"));
flowInArray[1].setNwSrc(flow.getJSONObject("match").getString("ipv4-source").substring(0, 8));
flowInArray[1].setNwDst(flow.getJSONObject("match").getString("ipv4-destination").substring(0, 8));
```

Array:

```java
{
  "flow-node-inventory:table": [
```
"id":0,
"flow-hash-id-map": [ (...)
 (...)
 (...)

"flow": [
 {
 "id":"S1:H4-H1:InicialFlowInsert",
 "instructions":{
 (...)
 (...)
 }
 }
 {
 "id":"ICMP:from:10.0.0.1:dest:10.0.0.3",
 "instructions":{
 "instruction":[
 {
 "order":0,
 "apply-actions":{
 "action":[
 {
 "order":0,
 "output-action":{
 "output-node-connector":"3",
 "max-length":0
 }
 },
 {
 "order":1,
 "output-action":{
 "output-node-connector":"5",
 "max-length":0
 }
 }
 ]
 }
 }
 }
 ]
 },
 "cookie":0,
 "match":{
 "ethernet-match":{
 "ethernet-type":{
 "type":2048
 }
 },
 "ipv4-source":"10.0.0.1/32",
 "ipv4-destination":"10.0.0.3/32"
Once we have filled the dictionary with all the installed flows, the External Orchestrator return to its main function (17, 18, 19).

6.4.14 Installing a new L3 definitive flow and waiting for the next one (19-23)

Aforementioned, the repetition control can be made by consulting to OpenDaylight database with a Rest call (14-18) or also creating our own database in the External Orchestrator. Both options are available in the GUI User Interface and if the user chose the second one, then it is the moment to create the dictionary. This dictionary will be exactly like the previous one, using Flow Model, but in this case we directly create a Flow Model dictionary and we inspect it with a loop (AppendixXX). In the case of having the incoming flow in the dictionary, it is rejected and an echo is printed in the GUI User Interface screen. In the opposite case, the flow is installed in the controller, the dictionary is updated and a message is printed at the User Interface screen. It has to be mentioned that, in order to install the new flow, the External Orchestrator repeats the procedure already made with the previous flows:
20) Creates the body of the new flow with JSON with the Rest Interface JSON Creator.

21) Establishes the connection with the OpenDaylight REST-API and send the JSON flow with the Rest Interface Sender.

23) Listens again the DPI with the Stock Socket Quote Server waiting for the next flow.
6. The External Orchestrator: Intelligent Outside App
Conclusions

In this project we have presented the necessity to change the traditional networks because of the need to increase efficiency in telecommunication. We have seen that due to stagnancy problems in traditional networks, the solution for this situation is not to physically enlarge the present infrastructures. For this reason, we have exposed a new possibility by the hand of the new paradigms offered to the Network Telecommunication Operators – Network Function Virtualization, Cloud Computing and Software Defined Networking –. We have explored and described these technologies in order to deeply enquire into OpenDaylight.

OpenDaylight is a novel and powerful controller that can give this aforementioned characteristics to the networks. First of all, we have deeply studied this controller in order to simulate different possible scenarios. The way to proceed has been to program the intelligence using all the possibilities that this controller offers, taking special attention which is the most appropriate for the network in each situation. The methodology used is the following:

- We have controlled our scenario by implementing several bundles inside the controller.
- We have implemented the control of our scenario from outside with another program (an External Orchestrator) and used OpenDaylight to translate the information to the network.
- We have also developed a hybrid solution in which we have programmed the
intelligence of the network externally but also we have taken advantage of some bundles already implemented in OpenDaylight.

After these extensive research and the corresponding implementations, we can conclude that OpenDaylight is a potent tool to program networks so it offers these three different possibilities depending on the specific user requires.

We can assert that if we want to control a network with an isolated controller (without any contact with other external tools) the easiest way is to implement a bundle inside OpenDaylight (Chapter 4). We have obtained consistent results according to the theoretical basis. Thus, in our scenarios, after the ARP packets have generated the route tables in the switches through the controller, we have managed to transmit the packets below milliseconds.

However, this architecture is becoming less popular: it is difficult to imagine a company or a network developer implementing its own bundle now. This behavior is due to the wish to contact with other several tools and devices outside to the controller (Chapter 5, 6). This technology limits a bit this will (Annotation and Reflections, page 32). So today, network developers mainly develop their External Orchestrators to manage the scenarios (Chapter 6).

After developing our own orchestrator, we have obtained the following conclusions:

OpenDaylight is a very powerful tool to translate the intelligence that we can create outside the controller (anywhere and with as many programs and devices as we need). In our case we have used an External Orchestrator and a Data Packet Inspector (DPI). The power falls on the Model Generated Rest API than provides the controller. This API gives to the user a vast range of interfaces to contact from outside the different functionalities of OpenDaylight. For example, to control the flows of our scenario in the chapter 6, we have used the inventory interface.

The obtained results have been clarifying: the firsts packets arriving from the
Internet to our network (scenario Chapter 6.1) have been analyzed by the DPI and the External Orchestrator, with a delay of around ten milliseconds. However, after installing the flows in the corresponding switches, all the packets have been transmitted below the millisecond.

It has to be explained that all the simulations in this dissertation have been virtual: we have used virtual nodes and hosts using Mininet (Chapter 2.5). However, nothing makes us think that the behavior would not be similar with physical devices.

To conclude, stress that OpenDaylight is a young controller with its corresponding bugs, not many documentation (but quickly growing) and very changeable: for example, after we had finished the ‘Rest Interface JSON Creator (Chapter 6.4.5)’ and the ‘Rest Interface Sender (Chapter 6.4.6)’, OpenDaylight changed completely its current REST API, which has entailed 2-3 weeks of changes including understanding the new interface without documentation. Anyway, we managed to overcome this issue and we end up with the External Orchestrator reaching all the functionalities we expected before beginning to develop it.
Appendix A

Code Developed
A.1 Mininet Topology (Python script): Two directly connected switches plus a host for each switch

host — switch — switch — host

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
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

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

topos = { 'mytopo': ( lambda: MyTopo() ) }
A.2 Mininet Topology (Python script): Specific Loop Topology

```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
        leftHosts1 = self.addHost('h1')
        rightHosts1 = self.addHost('h2')
        leftHosts2 = self.addHost('h3')
        rightHosts2 = self.addHost('h4')
        leftSwitch = self.addSwitch('s1')
        rightSwitch = self.addSwitch('s2')
        filterSwitch = self.addSwitch('s3')

        # Add links
        self.addLink(leftHosts1, leftSwitch)
        self.addLink(rightHosts1, leftSwitch)
        self.addLink(leftHosts2, rightSwitch)
        self.addLink(rightHosts2, rightSwitch)
        self.addLink(leftSwitch, rightSwitch)
        self.addLink(leftSwitch, filterSwitch)
        self.addLink(filterSwitch, rightSwitch)

    topos = { 'mytopo': ( lambda: MyTopo() ) }
```
A.3 ADSAL Hub

```java
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.sal.packet.Ethernet;
import org.opendaylight.controller.sal.packet.IDataPacketService;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.Packet;
import org.opendaylight.controller.sal.packet.PacketResult;
import org.opendaylight.controller.sal.packet.RawPacket;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.framework.FrameworkUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class AdsalHub implements IListenDataPacket {
    private static final Logger logger = LoggerFactory.getLogger(TutorialL2Forwarding.class);
    private ISwitchManager switchManager = null;
    private IFlowProgrammerService programmer = null;
    private IDataPacketService dataPacketService = null;

    private final Map<Long, NodeConnector> mac_to_port = new HashMap<Long, NodeConnector>();
    private final String function = "hub";

    void setDataPacketService(IDataPacketService s) {
        this.dataPacketService = s;
    }

    void unsetDataPacketService(IDataPacketService s) {
        if (this.dataPacketService == s) {
            this.dataPacketService = null;
        }
    }

    public void setFlowProgrammerService(IFlowProgrammerService s) {
```
```java
this.programmer = s;
}

public void unsetFlowProgrammerService(IFlowProgrammerService s) {
    if (this.programmer == s) {
        this.programmer = null;
    }
}

void setSwitchManager(ISwitchManager s) {
    logger.debug("SwitchManager set");
    this.switchManager = s;
}

void unsetSwitchManager(ISwitchManager s) {
    if (this.switchManager == s) {
        logger.debug("SwitchManager removed!");
        this.switchManager = null;
    }
}

/**
* Function called by the dependency manager when all the required
* dependencies are satisfied
* */
void init() {
    logger.info("Initialized");
    // Disabling the SimpleForwarding and ARPHandler bundle to not conflict
    // with this one
    BundleContext bundleContext = FrameworkUtil.getBundle(this.getClass()).
        getBundleContext();
    for (Bundle bundle : bundleContext.getBundles()) {
        if (bundle.getSymbolicName().contains("simpleforwarding")) {
            try {
                bundle.uninstall();
            } catch (BundleException e) {
                logger.error("Exception in Bundle uninstall "+bundle.
                    getSymbolicName(), e);
            }
        }
    }
}

/**
* Function called by the dependency manager when at least one dependency
* become unsatisfied or when the component is shutting down because for
* example bundle is being stopped.
* */
```
```java
void destroy() {
}
/**
 * Function called by dependency manager after "init ()" is called and after
 * the services provided by the class are registered in the service registry
 */
void start() {
    logger.info("Started");
}
/**
 * Function called by the dependency manager before the services exported by
 * the component are unregistered, this will be followed by a "destroy ()"
 * calls
 */
void stop() {
    logger.info("Stopped");
}
private void floodPacket(RawPacket inPkt) {
    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    // Get ports on the switch where the packet arrived
    Set<NodeConnector> nodeConnectors = this.switchManager
        .getUpNodeConnectors(incoming_node);

    for (NodeConnector p : nodeConnectors) {
        if (!p.equals(incoming_connector)) {
            try {
                RawPacket destPkt = new RawPacket(inPkt);
                destPkt.setOutgoingNodeConnector(p);
                this.dataPacketService.transmitDataPacket(destPkt);
            } catch (ConstructionException e2) {
                continue;
            }
        }
    }
}
@Override
public PacketResult receiveDataPacket(RawPacket inPkt) {
    if (inPkt == null) {
        return PacketResult.IGNORED;
    }
    logger.trace("Received a frame of size: {}",
        inPkt.getPacketData().length);
```
// Hub implementation

floodPacket(inPkt);

return PacketResult.IGNORED;
}
A.4 ADSAL Bundle Activator

```java
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.util.Hashtable;
import java.util.Dictionary;
import org.apache.felix.dm.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.opendaylight.controller.sal.core.ComponentActivatorAbstractBase;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.IDataPacketService;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.switchmanager.ISwitchManager;

public class Activator extends ComponentActivatorAbstractBase {

  protected static final Logger logger = LoggerFactory.getLogger(Activator.class);

  /**
   * Function called when the activator starts just after some initializations are done by the
   * ComponentActivatorAbstractBase.
   *
   */
  public void init() {
  }

  /**
   * Function called when the activator stops just before the cleanup done by ComponentActivatorAbstractBase
   *
   */
  public void destroy() {
  }

  /**
   * Function that is used to communicate to dependency manager the list of known implementations for services inside a container
   *
   * @return An array containing all the CLASS objects that will be instantiated in order to get an fully working implementation
   * Object
   */
  }
public Object[] getImplementations() {
    Object[] res = { TutorialL2Forwarding.class }; 
    return res; 
}

/**
 * Function that is called when configuration of the dependencies
 * is required.
 * @param c dependency manager Component object, used for
 * configuring the dependencies exported and imported
 * @param imp Implementation class that is being configured,
 * needed as long as the same routine can configure multiple
 * implementations
 * @param containerName The containerName being configured, this allow
 * also optional per-container different behavior if needed, usually
 * should not be the case though.
 */
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));

        c.add(createContainerServiceDependency(containerName).setService(
            IDataPacketService.class).setCallbacks(
            "setDataPacketService", "unsetDataPacketService")
            .setRequired(true));

        c.add(createContainerServiceDependency(containerName).setService(
            IFlowProgrammerService.class).setCallbacks(
            "setFlowProgrammerService", "unsetFlowProgrammerService")
            .setRequired(true));
    }
}
A.5 ADSAL Learning SINGLE Switch

```
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.lang.String;
import java.util.Map;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.ConcurrentHashMap;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.framework.FrameworkUtil;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.sal.packet.ARP;
import org.opendaylight.controller.sal.packet.BitBufferHelper;
import org.opendaylight.controller.sal.packet.Ethernet;
import org.opendaylight.controller.sal.packet.ICMP;
import org.opendaylight.controller.sal.packet.IDataPacketService;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.Packet;
import org.opendaylight.controller.sal.packet.PacketResult;
import org.opendaylight.controller.sal.packet.RawPacket;
import org.opendaylight.controller.sal.action.Action;
import org.opendaylight.controller.sal.action.Output;
import org.opendaylight.controller.sal.action.Flood;
import org.opendaylight.controller.sal.match.Match;
import org.opendaylight.controller.sal.match.MatchType;
import org.opendaylight.controller.sal.match.MatchField;
import org.opendaylight.controller.sal.utils.EtherTypes;
import org.opendaylight.controller.sal.utils.Status;
```
import org.opendaylight.controller.sal.utils.NetUtils;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.opendaylight.controller.switchmanager.Subnet;

public class AdsalLearninSingleSwitch implements IListenDataPacket {
    private static final Logger logger = LoggerFactory.getLogger(TutorialL2Forwarding.class);
    private ISwitchManager switchManager = null;
    private IFlowProgrammerService programmer = null;
    private IDataPacketService dataPacketService = null;
    private Map<Long, NodeConnector> mac_to_port = new HashMap<Long, NodeConnector>();
    private String function = "hub";

    void setDataPacketService(IDataPacketService s) {
        this.dataPacketService = s;
    }

    void unsetDataPacketService(IDataPacketService s) {
        if (this.dataPacketService == s) {
            this.dataPacketService = null;
        }
    }

    public void setFlowProgrammerService(IFlowProgrammerService s) {
        this.programmer = s;
    }

    public void unsetFlowProgrammerService(IFlowProgrammerService s) {
        if (this.programmer == s) {
            this.programmer = null;
        }
    }

    void setSwitchManager(ISwitchManager s) {
        logger.debug("SwitchManager set");
        this.switchManager = s;
    }

    void unsetSwitchManager(ISwitchManager s) {
        if (this.switchManager == s) {
            logger.debug("SwitchManager removed!");
            this.switchManager = null;
        }
    }

    /**
     * Function called by the dependency manager when all the required
void init() {
    logger.info("Initialized");
    // Disabling the SimpleForwarding and ARPHandler bundle to not conflict
    // with this one
    BundleContext bundleContext = FrameworkUtil.getBundle(this.getClass()).
        getBundleContext();
    for(Bundle bundle : bundleContext.getBundles()) {
        if (bundle.getSymbolicName().contains("simpleforwarding")) {
            try {
                bundle.uninstall();
            } catch (BundleException e) {
                logger.error("Exception in Bundle uninstall "+bundle.
                    getSymbolicName(), e);
            }
        }
    }
}

/**
 * Function called by the dependency manager when at least one
 * dependency become unsatisfied or when the component is shutting
 * down because for example bundle is being stopped.
 */
void destroy() {
}

 /**
 * Function called by dependency manager after "init ()" is called
 * and after the services provided by the class are registered in
 * the service registry
 */
void start() {
    logger.info("Started");
}

 /**
 * Function called by the dependency manager before the services
 * exported by the component are unregistered, this will be
 * followed by a "destroy ()" calls
 */
void stop() {
    logger.info("Stopped");
}
private void floodPacket(RawPacket inPkt) {
    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    Set<NodeConnector> nodeConnectors =
            this.switchManager.getUpNodeConnectors(incoming_node);

    for (NodeConnector p : nodeConnectors) {
        if (!p.equals(incoming_connector)) {
            try {
                RawPacket destPkt = new RawPacket(inPkt);
                destPkt.setOutgoingNodeConnector(p);
                this.dataPacketService.transmitDataPacket(destPkt);            
            } catch (ConstructionException e2) {
                continue;
            }
        }
    }
}

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

    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    NodeConnector dst_connector = null;
    // Extract packet from inPkt
    Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);

    if (!(formattedPak instanceof Ethernet)) {
        return PacketResult.IGNORED;
    }

    // Extract switch and port information (incoming_connector)
    // Extract packet srcMAC and dstMAC using getSourceMACAddress() and
// getDestinationMACAddress() of Ethernet class, and  
// convert to hex long we can use through BitBufferHelper  
long srcMAC_val = getSourceMACAddress(formattedPak);  
long dstMAC_val = getDestinationMACAddress(formattedPak);  

// Store srcMAC and incoming_connector in the mapping dictionary  
mac_to_port.put(srcMAC_val, incoming_connector);  

// Do I know the destination MAC? Lookup dstMAC in dictionary  
// If found, generate match and action for the flow, and program a  
// flow rule in switch  
boolean isDstMACKnown = mac_to_port.containsKey(dstMAC_val);  

if (isDstMACKnown) {  
dst_connector = mac_to_port.get(dstMAC_val);  

List<Action> actions = new ArrayList<Action>();  
actions.add(new Output(dst_connector));  

byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();  

Match match = new Match();  
match.setField( new MatchField(MatchType.IN_PORT, incoming_connector) );  
match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) );  

Flow f = new Flow(match, actions);  

// 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;  
} else {  
    try {  
inPkt.setOutgoingNodeConnector(dst_connector);  
    this.dataPacketService.transmitDataPacket(inPkt);  
    } catch (Exception e) {  
        logger.info("Error transmitting the Packet");  
    }  
    logger.info("Installed flow {} in node {}", f, incoming_node);  
} else {  
    floodPacket(inPkt);  
}  

return PacketResult.CONSUME;
private long getSourceMACAddress(Packet formattedPak) {
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
    long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    return srcMAC_val;
}

private long getDestinationMACAddress(Packet formattedPak) {
    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
    return dstMAC_val;
}
A.6 ADSAL Learning MULTI Switch

```java
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.ConcurrentHashMap;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.sal.flowprogrammer.Flow;
import org.opendaylight.controller.sal.packet.ARP;
import org.opendaylight.controller.sal.packet.BitBufferHelper;
import org.opendaylight.controller.sal.packet.Ethernet;
import org.opendaylight.controller.sal.packet.ICMP;
import org.opendaylight.controller.sal.packet.IDataPacketService;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.Packet;
import org.opendaylight.controller.sal.packet.PacketResult;
import org.opendaylight.controller.sal.packet.RawPacket;
import org.opendaylight.controller.sal.action.Action;
import org.opendaylight.controller.sal.action.Output;
import org.opendaylight.controller.sal.action.Flood;
import org.opendaylight.controller.sal.match.Match;
import org.opendaylight.controller.sal.match.MatchType;
import org.opendaylight.controller.sal.match.MatchField;
import org.opendaylight.controller.sal.utils.EtherTypes;
import org.opendaylight.controller.sal.utils.Status;
```
A.6 ADSAL Learning MULTI Switch

import org.opendaylight.controller.sal.utils.NetUtils;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.opendaylight.controller.switchmanager.Subnet;

public class AdsalLearningMultiSwitch implements IListenDataPacket {
    private static final Logger logger = LoggerFactory.getLogger(TutorialL2Forwarding.class);
    private ISwitchManager switchManager = null;
    private IFlowProgrammerService programmer = null;
    private IDataPacketService dataPacketService = null;
    private Map<Node, Map<Long, NodeConnector>> mac_to_port_per_switch = new HashMap<Node, Map<Long, NodeConnector>>();
    private String function = "hub";

    void setDataPacketService(IDataPacketService s) {
        this.dataPacketService = s;
    }

    void unsetDataPacketService(IDataPacketService s) {
        if (this.dataPacketService == s) {
            this.dataPacketService = null;
        }
    }

    public void setFlowProgrammerService(IFlowProgrammerService s) {
        this.programmer = s;
    }

    public void unsetFlowProgrammerService(IFlowProgrammerService s) {
        if (this.programmer == s) {
            this.programmer = null;
        }
    }

    void setSwitchManager(ISwitchManager s) {
        logger.debug("SwitchManager set");
        this.switchManager = s;
    }

    void unsetSwitchManager(ISwitchManager s) {
        if (this.switchManager == s) {
            logger.debug("SwitchManager removed!");
            this.switchManager = null;
        }
    }

    /**
     * Function called by the dependency manager when all the required
void init() {
    logger.info("Initialized");
    // Disabling the SimpleForwarding and ARPHandler bundle to not conflict
    // with this one
    BundleContext bundleContext = FrameworkUtil.getBundle(this.getClass()).
        getBundleContext();
    for(Bundle bundle : bundleContext.getBundles()) {
        if (bundle.getSymbolicName().contains("simpleforwarding")) {
            try {
                bundle.uninstall();
            } catch (BundleException e) {
                logger.error("Exception in Bundle uninstall ", bundle.
                    getSymbolicName(), e);
            }
        }
    }
}

/**
 * Function called by the dependency manager when at least one
 * dependency become unsatisfied or when the component is shutting
 * down because for example bundle is being stopped.
 */
void destroy() {
}

/**
 * Function called by dependency manager after "init ()" is called
 * and after the services provided by the class are registered in
 * the service registry
 */
void start() {
    logger.info("Started");
}

/**
 * Function called by the dependency manager before the services
 * exported by the component are unregistered, this will be
 * followed by a "destroy ()" calls
 */
void stop() {
    logger.info("Stopped");
private void floodPacket(RawPacket inPkt) {
    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    Set<NodeConnector> nodeConnectors =
        this.switchManager.getUpNodeConnectors(incoming_node);

    for (NodeConnector p : nodeConnectors) {
        if (!p.equals(incoming_connector)) {
            try {
                RawPacket destPkt = new RawPacket(inPkt);
                destPkt.setOutgoingNodeConnector(p);
                this.dataPacketService.transmitDataPacket(destPkt);
            } catch (ConstructionException e2) {
                continue;
            }
        }
    }
}

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

    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    // Extract packet from inPkt
    Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);

    if (!(formattedPak instanceof Ethernet)) {
        return PacketResult.IGNORED;
    }

    // Extract switch and port information (incoming_connector)

    // Extract packet srcMAC and dstMAC using getSourceMACAddress() and
    // getDestinationMACAddress() of Ethernet class, and
    // convert to hex long we can use through BitBufferHelper
```java
long srcMAC_val = getSourceMACAdress(formattedPak);
long dstMAC_val = getDestinationMACAdress(formattedPak);

// Store srcMAC and incoming_connector in the mapping dictionary of each
// DIFFERENT SWITCH
if(!mac_to_port_per_switch.containsKey(incoming_node)) {
    mac_to_port_per_switch.put(incoming_node, new HashMap<Long,
    NodeConnector>());
}
mac_to_port_per_switch.get(incoming_node).put(srcMAC_val,
    incoming_connector);

// Do I know the destination MAC? Lookup dstMAC in dictionary
// If found, generate match and action for the flow, and program a
// flow rule in switch
boolean isDstMACKnown = mac_to_port_per_switch.get(incoming_node).
    containsKey(dstMAC_val);

if (isDstMACKnown) {
    NodeConnector dst_connector = mac_to_port_per_switch.get(incoming_node).
        get(dstMAC_val);

    List<Action> actions = new ArrayList<Action>();
    actions.add(new Output(dst_connector));

    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();

    Match match = new Match();
    match.setField( new MatchField(MatchType.IN_PORT, incoming_connector) );
    match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) );

    Flow f = new Flow(match, actions);
    f.setPriority((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;
    } else {
        try {
            inPkt.setOutgoingNodeConnector(dst_connector);
            this.dataPacketService.transmitDataPacket(inPkt);
        } catch (Exception e) {
            logger.info("Error transmitting the Packet");
        }
    }
```

logger.info("Installed flow {} in node {}", f, incoming_node);
}
else {
    floodPacket(inPkt);
}

return PacketResult.CONSUME;

private long getSourceMACAddress(Packet formattedPak) {
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
    long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    return srcMAC_val;
}

private long getDestinationMACAddress(Packet formattedPak) {
    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
    return dstMAC_val;
}
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Set;
import java.util.HashSet;
import java.lang.String;
import java.util.Map;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.ConcurrentHashMap;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.sal.flowprogrammer.Flow;
import org.opendaylight.controller.sal.packet.ARP;
import org.opendaylight.controller.sal.packet.BitBufferHelper;
import org.opendaylight.controller.sal.packet.Ethernet;
import org.opendaylight.controller.sal.packet.ICMP;
import org.opendaylight.controller.sal.packet.IpDataPacketService;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.Packet;
import org.opendaylight.controller.sal.packet.PacketResult;
import org.opendaylight.controller.sal.packet.RawPacket;
import org.opendaylight.controller.sal.action.Action;
import org.opendaylight.controller.sal.action.Output;
import org.opendaylight.controller.sal.action.Flood;
import org.opendaylight.controller.sal.match.Match;
import org.opendaylight.controller.sal.match.MatchType;
import org.opendaylight.controller.sal.match.MatchField;
import org.opendaylight.controller.sal.utils.EtherTypes;
import org.opendaylight.controller.sal.utils.Status;
import org.opendaylight.controller.sal.utils.NetUtils;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.opendaylight.controller.switchmanager.Subnet;

public class AdsalSpecificTopology implements IListenDataPacket {
    private static final Logger logger = LoggerFactory.getLogger(TutorialL2Forwarding.class);
    private ISwitchManager switchManager = null;
    private IFlowProgrammerService programmer = null;
    private IDataPacketService dataPacketService = null;
    private Map<Node, Map<Long, Map<Long, NodeConnector>>> mac_to_port_per_switch_and_dest = new HashMap<Node, Map<Long, Map<Long, NodeConnector>>>();
    private String function = "switch";

    void setDataPacketService(IDataPacketService s) {
        this.dataPacketService = s;
    }

    void unsetDataPacketService(IDataPacketService s) {
        if (this.dataPacketService == s) {
            this.dataPacketService = null;
        }
    }

    public void setFlowProgrammerService(IFlowProgrammerService s) {
        this.programmer = s;
    }

    public void unsetFlowProgrammerService(IFlowProgrammerService s) {
        if (this.programmer == s) {
            this.programmer = null;
        }
    }

    void setSwitchManager(ISwitchManager s) {
        logger.debug("SwitchManager set");
        this.switchManager = s;
    }

    void unsetSwitchManager(ISwitchManager s) {
        if (this.switchManager == s) {
            logger.debug("SwitchManager removed!");
            this.switchManager = null;
        }
    }

    /**
     * Function called by the dependency manager when all the required
     */
}
void init() {
    logger.info("Initialized");
    // Disabling the SimpleForwarding and ARPHandler bundle to not
    // conflict with this one
    BundleContext bundleContext = FrameworkUtil.getBundle(this.getClass()).
    getBundleContext();
    for (Bundle bundle : bundleContext.getBundles()) {
        if (bundle.getSymbolicName().contains("simpleforwarding")) {
            try {
                bundle.uninstall();
            } catch (BundleException e) {
                logger.error("Exception in Bundle uninstall " +
                bundle.getSymbolicName(), e);
            }
        }
    }
}

/**
 * Function called by the dependency manager when at least one
 * dependency become unsatisfied or when the component is shutting
 * down because for example bundle is being stopped.
 */
void destroy() {
}

/**
 * Function called by dependency manager after "init ()" is called
 * and after the services provided by the class are registered in
 * the service registry
 */
void start() {
    logger.info("Started");
}

/**
 * Function called by the dependency manager before the services
 * exported by the component are unregistered, this will be
 * followed by a "destroy ()" calls
 */
void stop() {
    logger.info("Stopped");
}
private void floodPacket(RawPacket inPkt) {
    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    Set<NodeConnector> nodeConnectors =
        this.switchManager.getUpNodeConnectors(incoming_node);

    for (NodeConnector p : nodeConnectors) {
        Object id = p.getID();
        int id_num = (int) id;

        if (!p.equals(incoming_connector) && id_num!=3) {
            try {
                RawPacket destPkt = new RawPacket(inPkt);
                destPkt.setOutgoingNodeConnector(p);
                this.dataPacketService.transmitDataPacket(destPkt);
            } catch (ConstructionException e2) {
                continue;
            }
        }
    }
}

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

    // Extract switch and port information (incoming_connector)
    NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
    Node incoming_node = incoming_connector.getNode();

    // Hub implementation
    if (function.equals("hub")){
        floodPacket(inPkt);
        return PacketResult.CONSUME;
    }
Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);

if (!(formattedPak instanceof Ethernet)) {
    return PacketResult.IGNORED;
}

long srcMAC_val = getSourceMACAddress(formattedPak);
long dstMAC_val = getDestinationMACAddress(formattedPak);

if(srcMAC_val<100000 && dstMAC_val<100000){
    Set<NodeConnector> nodeConnectors =
        this.switchManager.getUpNodeConnectors(incoming_node);

    NodeConnector nodeConnectors_array[] = new NodeConnector[4];
    nodeConnectors.toArray(nodeConnectors_array);

    Object actualSwitch = incoming_node.getID();
    int actualSwitch_val = (int)(long) actualSwitch;

    List<Action> actions = new ArrayList<Action>();

    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();

    Match match = new Match();
    match.setField( new MatchField(MatchType.IN_PORT, incoming_connector) );
    match.setField( new MatchField(MatchType.DL_SRC, srcMAC.clone()) );
    match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) );

    switch(actualSwitch_val){
        case 1:
            switch((int)(long)srcMAC_val){
                case 1:
                    switch((int)(long)dstMAC_val){
                        case 2:
                            inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
                            actions.add(new Output(nodeConnectors_array[2]));
                            break;
                        case 3:
                            inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
                            break;
                    }
                case 3:
                    break;
            }
        case 2:
            break;
    }
}
actions.add(new Output(nodeConnectors_array[1]));
break;
case 4:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
actions.add(new Output(nodeConnectors_array[0]));
break;
}
break;
case 2:
switch((int)(long)dstMAC_val){
case 1:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
actions.add(new Output(nodeConnectors_array[3]));
break;
case 3:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
actions.add(new Output(nodeConnectors_array[0]));
break;
case 4:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
actions.add(new Output(nodeConnectors_array[1]));
break;
}
break;
case 3:
switch((int)(long)dstMAC_val){
case 1:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
actions.add(new Output(nodeConnectors_array[3]));
break;
case 2:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
actions.add(new Output(nodeConnectors_array[2]));
break;
}
break;
case 4:
switch((int)(long)dstMAC_val){
case 1:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
actions.add(new Output(nodeConnectors_array[3]));
break;
case 2:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
actions.add(new Output(nodeConnectors_array[2]));
break;
}
break;
break;

case 2:
    switch((int)(long)srcMAC_val){
    case 1:
        switch((int)(long)dstMAC_val){
        case 3:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
            actions.add(new Output(nodeConnectors_array[3]));
            break;
        case 4:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
            actions.add(new Output(nodeConnectors_array[2]));
            break;
        }
        break;
    case 2:
        switch((int)(long)dstMAC_val){
        case 3:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
            actions.add(new Output(nodeConnectors_array[3]));
            break;
        case 4:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
            actions.add(new Output(nodeConnectors_array[2]));
            break;
        }
        break;
    case 3:
        switch((int)(long)dstMAC_val){
        case 1:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
            actions.add(new Output(nodeConnectors_array[1]));
            break;
        case 2:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
            actions.add(new Output(nodeConnectors_array[0]));
            break;
        case 4:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
            actions.add(new Output(nodeConnectors_array[2]));
            break;
        }
        break;
    case 4:
        switch((int)(long)dstMAC_val){
        case 1:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
            actions.add(new Output(nodeConnectors_array[0]));
            break;
        case 3:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
            actions.add(new Output(nodeConnectors_array[3]));
            break;
        case 4:
            inPkt.setOutgoingNodeConnector(nodeConnectors_array[2]);
            actions.add(new Output(nodeConnectors_array[2]));
            break;
        }
        break;
break;
case 2:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
actions.add(new Output(nodeConnectors_array[1]));
break;
case 3:
inPkt.setOutgoingNodeConnector(nodeConnectors_array[3]);
actions.add(new Output(nodeConnectors_array[3]));
break;
}
break;

case 3:
switch((int)(long)srcMAC_val){
case 1:
  if(dstMAC_val==4){
    inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
    actions.add(new Output(nodeConnectors_array[0]));
  }
  break;
case 2:
  if(dstMAC_val==3){
    inPkt.setOutgoingNodeConnector(nodeConnectors_array[0]);
    actions.add(new Output(nodeConnectors_array[0]));
  }
  break;
case 3:
  if(dstMAC_val==2){
    inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
    actions.add(new Output(nodeConnectors_array[1]));
  }
  break;
case 4:
  if(dstMAC_val==1){
    inPkt.setOutgoingNodeConnector(nodeConnectors_array[1]);
    actions.add(new Output(nodeConnectors_array[1]));
  }
  break;
}

default:
}

Flow f = new Flow(match, actions);
f.setPriority((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;
} else {
    try {
        this.dataPacketService.transmitDataPacket(inPkt);
    } catch (Exception e) {
        logger.info("Error transmitting the Packet");
    }
    logger.info("Installed flow {} in node {}", f, incoming_node);
} else {
    floodPacket(inPkt);
}
return PacketResult.CONSUME;

private long getSourceMACAdress(Packet formattedPak) {
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
    long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    return srcMAC_val;
}

private long getDestinationMACAdress(Packet formattedPak) {
    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
    return dstMAC_val;
}
A.8 Mininet Topology (Python script): External Orchestrator Scenario

```python
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink

# Defining the network
def myNetwork():
    net = Mininet(topo=None,
                  build=False, link=TCLink)
    info('*** Adding controller
') # The info command enable
    net.addController(name='c0',
                       controller=RemoteController,
                       # ip='192.168.224.133',
                       port=6633)

    # Adding the switches
    info('*** Add switches
')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    # Adding the hosts
    info('*** Add hosts
')
    h1 = net.addHost('h1', ip='10.0.0.1')
    h2 = net.addHost('h2', ip='10.0.0.2')
    h3 = net.addHost('h3', ip='10.0.0.3')
    h4 = net.addHost('h4', ip='10.0.0.4')

    # Adding the physical connections between switches and hosts
    info('*** Add links
')
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s2)
    net.addLink(h4, s2)
    net.addLink(s1, s2)
    net.addLink(s1, s3)
```

A.8 Mininet Topology (Python script): External Orchestrator Scenario
net.addLink(s3, s2)

Intf('eth2', node = s1)

info('*** Starting network\n')

# Building the network
net.start()

# Adding a connection (port) between the switch s1 and the
# eth1 port of the Virtual Machine
s1.cmd('ovs-vsctl add-port s1 eth2')
# s1.cmd('ifconfig s1 10.0.0.5')

# Creating the CLI for the network
CLI(net)
net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    myNetwork()
public class GUIUserInterface {
    private static JFrame frame;
    private JTextField textField;
    private JTextField textField_1;
private JTextField textField_2;
private JTextField textField_3;
private JTextField textField_4;
private JTextField textField_5;
private JTextField textField_6;
private JTextField textField_7;
private JTextField textField_8;
private JTextField textField_9;
private JTextField textField_10;
private JTextField textField_11;
private JTextField textField_12;
private JTextField textField_13;
private JTextField textField_14;
private JTextField textField_15;
private JTextField textField_16;
private JTextField textField_17;
private JTextField textField_18;
private JTextField textField_19;

/**
 * @wbp.nonvisual location=-28,-23
 */

/**
 * Launch the application.
 */

public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                GUIUserInterface window = new GUIUserInterface();
                window.frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the application.
 */

public GUIUserInterface() {
    initialize();
}

/**
 * Initialize the contents of the frame.
 */

private void initialize() {
frame = new JFrame();   
frame.setBounds(100, 0, 1366, 711);   
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   
frame.getContentPane().setLayout(null);   

JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);   
tabbedPane.setBounds(1, 1, 1377, 779);   
Image img = new ImageIcon(this.getClass().getResource("/mytopo.png")).getImage();

frame.getContentPane().add(tabbedPane);

// First Panel  
JPanel panel = new JPanel();   
panel.setLayout(null);   

JButton btnRun = new JButton("RUN");   
btnRun.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        ExternalOrchestrator initApp = new ExternalOrchestrator();
        try {
            initApp.FullAplication();
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});   
btnRun.setBounds(1178, 59, 117, 25);   
panel.add(btnRun);

JRadioButton rdbtnHelium = new JRadioButton("HELIUM & LITHIUM");   
rdbtnHelium.setBackground(Color.lightGray);   
rdbtnHelium.setBounds(284, 60, 163, 23);   
panel.add(rdbtnHelium);

JLabel topology = new JLabel("");   
topology.setBounds(722, 151, 573, 483);   
topology.setIcon(new ImageIcon(img));   
topology.setBorder(BorderFactory.createMatteBorder(20, 20, 20, 20, Color.LIGHT_GRAY));   
panel.add(topology);
JRadioButton rdbtnHidrogen = new JRadioButton("HIDROGEN");
rdbtnHidrogen.setBackground(Color.lightGray);
rdbtnHidrogen.setBounds(140, 60, 163, 23);
panel.add(rdbtnHidrogen);

JRadioButton rdbtnRest = new JRadioButton("REST");
rdbtnRest.setBackground(Color.lightGray);
rdbtnRest.setBounds(817, 60, 163, 23);
panel.add(rdbtnRest);

JRadioButton rdbtnOwnDictionary = new JRadioButton("OWN DICTIONARY");
rdbtnOwnDictionary.setBackground(Color.lightGray);
rdbtnOwnDictionary.setBounds(965, 60, 163, 23);
panel.add(rdbtnOwnDictionary);

JLabel lblRelease = new JLabel("RELEASE");
lblRelease.setForeground(Color.WHITE);
lblRelease.setBackground(Color.lightGray);
lblRelease.setFont(new Font("Dialog", Font.BOLD, 16));
lblRelease.setBounds(38, 61, 75, 19);
panel.add(lblRelease);

JLabel DupControl_prova = new JLabel("Duplication Control");
DupControl_prova.setForeground(Color.WHITE);
DupControl_prova.setBackground(Color.lightGray);
DupControl_prova.setFont(new Font("Dialog", Font.BOLD, 16));
DupControl_prova.setBounds(624, 61, 168, 19);
panel.add(DupControl_prova);

JLabel marc_prova = new JLabel("";
marc_prova.setOpaque(true);
marc_prova.setForeground(Color.LIGHT_GRAY);
marc_prova.setBackground(Color.lightGray);
marc_prova.setBounds(12, 46, 1334, 49);
panel.add(marc_prova);

tabbedPane.addTab("ONICA", null, panel, null);

// Second Panel

JPanel panel_1 = new JPanel();
tabbedPane.addTab("Settings", null, panel_1, null);
panel_1.setLayout(null);

JLabel lblNewLabel = new JLabel("Username");
lblNewLabel.setBounds(824, 100, 100, 15);
lblNewLabel.setForeground(Color.WHITE);
panel_1.add(lblNewLabel);

textField = new JTextField();
textField.setBounds(929, 98, 114, 19);
textField.setColumns(10);
panel_1.add(textField);

JLabel lblPassword = new JLabel("Password");
lblPassword.setBounds(824, 144, 80, 15);
lblPassword.setForeground(Color.WHITE);
panel_1.add(lblPassword);

textField_1 = new JTextField();
textField_1.setColumns(10);
textField_1.setBounds(929, 142, 114, 19);
panel_1.add(textField_1);

JLabel label_2 = new JLabel("URL");
label_2.setBounds(824, 185, 70, 15);
label_2.setForeground(Color.WHITE);
panel_1.add(label_2);

textField_2 = new JTextField();
textField_2.setColumns(10);
textField_2.setBounds(929, 183, 114, 19);
panel_1.add(textField_2);

JLabel lblSocketserverListeningThe = new JLabel("SocketServer listening DPI");
lblSocketserverListeningThe.setBounds(824, 244, 202, 15);
lblSocketserverListeningThe.setForeground(Color.WHITE);
panel_1.add(lblSocketserverListeningThe);

JLabel lblNewLabel_1 = new JLabel("REST params to log");
lblNewLabel_1.setForeground(Color.WHITE);
lblNewLabel_1.setBounds(841, 63, 202, 15);
panel_1.add(lblNewLabel_1);

JLabel label_1 = new JLabel("@IP");
label_1.setForeground(Color.WHITE);
label_1.setBounds(824, 277, 100, 15);
panel_1.add(label_1);
textField_3 = new JTextField();
textField_3.setColumns(10);
textField_3.setBounds(929, 275, 114, 19);
panel_1.add(textField_3);

JLabel label_4 = new JLabel("Lithium & Helium params");
label_4.setForeground(Color.WHITE);
label_4.setBounds(824, 359, 202, 15);
panel_1.add(label_4);

textField_4 = new JTextField();
textField_4.setColumns(10);
textField_4.setBounds(929, 307, 114, 19);
panel_1.add(textField_4);

JLabel label_5 = new JLabel("Port");
label_5.setForeground(Color.WHITE);
label_5.setBounds(824, 309, 100, 15);
panel_1.add(label_5);

JLabel label_3 = new JLabel("Switch Table");
label_3.setForeground(Color.WHITE);
label_3.setBounds(824, 388, 100, 15);
panel_1.add(label_3);

textField_5 = new JTextField();
textField_5.setColumns(10);
textField_5.setBounds(929, 386, 114, 19);
panel_1.add(textField_5);

JLabel label_6 = new JLabel("Host Mask");
label_6.setForeground(Color.WHITE);
label_6.setBounds(824, 425, 100, 15);
panel_1.add(label_6);

textField_6 = new JTextField();
textField_6.setColumns(10);
textField_6.setBounds(929, 423, 114, 19);
panel_1.add(textField_6);

JLabel label_7 = new JLabel("@IP Host 3");
label_7.setForeground(Color.WHITE);
label_7.setBounds(1094, 148, 87, 15);
panel_1.add(label_7);

textField_7 = new JTextField();
textField_7.setColumns(10);
textField_7.setBounds(1199, 146, 114, 19);
A.9 GUI User Interface

```java
panel_1.add(textField_7);

JLabel label_8 = new JLabel("@IP Host 2" newX29 ;
    label_8.setForeground(Color.WHITE);
    label_8.setBounds(1094, 107, 80, 15);
    panel_1.add(label_8);

    textField_8 = new JTextField();
    textField_8.setColumns(10);
    textField_8.setBounds(1199, 105, 114, 19);
    panel_1.add(textField_8);

    JLabel label_9 = new JLabel("@IP Host 1" newX29 ;
    label_9.setForeground(Color.WHITE);
    label_9.setBounds(1094, 63, 100, 15);
    panel_1.add(label_9);

    textField_9 = new JTextField();
    textField_9.setColumns(10);
    textField_9.setBounds(1199, 61, 114, 19);
    panel_1.add(textField_9);

    JLabel label_10 = new JLabel("Switch 3 name" newX29 ;
    label_10.setForeground(Color.WHITE);
    label_10.setBounds(1094, 355, 114, 15);
    panel_1.add(label_10);

    textField_10 = new JTextField();
    textField_10.setColumns(10);
    textField_10.setBounds(1199, 353, 114, 19);
    panel_1.add(textField_10);

    JLabel label_11 = new JLabel("Switch 2 name" newX29 ;
    label_11.setForeground(Color.WHITE);
    label_11.setBounds(1094, 314, 114, 15);
    panel_1.add(label_11);

    textField_11 = new JTextField();
    textField_11.setColumns(10);
    textField_11.setBounds(1199, 312, 114, 19);
    panel_1.add(textField_11);

    JLabel label_12 = new JLabel("Switch 1 name" newX29 ;
    label_12.setForeground(Color.WHITE);
    label_12.setBounds(1094, 270, 114, 15);
    panel_1.add(label_12);

    textField_12 = new JTextField();
    textField_12.setColumns(10);
```
textField_12.setBounds(1199, 268, 114, 19);
panel_1.add(textField_12);

JLabel label_13 = new JLabel("Switch 1 Port 3");
lable_13.setForeground(Color.WHITE);
lable_13.setBounds(1094, 510, 114, 15);
panel_1.add(label_13);

textField_13 = new JTextField();
textField_13.setColumns(10);
textField_13.setBounds(1213, 508, 114, 19);
panel_1.add(textField_13);

JLabel label_14 = new JLabel("Switch 1 Port 2");
lable_14.setForeground(Color.WHITE);
lable_14.setBounds(1094, 469, 114, 15);
panel_1.add(label_14);

textField_14 = new JTextField();
textField_14.setColumns(10);
textField_14.setBounds(1213, 467, 114, 19);
panel_1.add(textField_14);

JLabel label_15 = new JLabel("Switch 1 Port 1");
lable_15.setForeground(Color.WHITE);
lable_15.setBounds(1094, 425, 114, 15);
panel_1.add(label_15);

textField_15 = new JTextField();
textField_15.setColumns(10);
textField_15.setBounds(1213, 423, 114, 19);
panel_1.add(textField_15);

JLabel label_16 = new JLabel("Switch 2 Port 3");
lable_16.setForeground(Color.WHITE);
lable_16.setBounds(1094, 645, 114, 15);
panel_1.add(label_16);

textField_16 = new JTextField();
textField_16.setColumns(10);
textField_16.setBounds(1213, 643, 114, 19);
panel_1.add(textField_16);

JLabel label_17 = new JLabel("Switch 2 Port 2");
lable_17.setForeground(Color.WHITE);
lable_17.setBounds(1094, 604, 114, 15);
panel_1.add(label_17);

textField_17 = new JTextField();
textField_17.setColumns(10);
textField_17.setBounds(1213, 602, 114, 19);
panel_1.add(textField_17);

JLabel label_18 = new JLabel("Switch 2 Port 1");
label_18.setForeground(Color.WHITE);
label_18.setBounds(1094, 560, 114, 15);
panel_1.add(label_18);

textField_18 = new JTextField();
textField_18.setColumns(10);
textField_18.setBounds(1213, 558, 114, 19);
panel_1.add(textField_18);

JLabel label_19 = new JLabel("@IP Host 4");
label_19.setForeground(Color.WHITE);
label_19.setBounds(1094, 189, 87, 15);
panel_1.add(label_19);

textField_19 = new JTextField();
textField_19.setColumns(10);
textField_19.setBounds(1199, 187, 114, 19);
panel_1.add(textField_19);

JLabel lblSettings = new JLabel("SETTINGS");
lblSettings.setFont(new Font("Droid Sans Mono", Font.BOLD, 33));
lblSettings.setForeground(Color.WHITE);
lblSettings.setBounds(105, 45, 202, 39);
panel_1.add(lblSettings);

JRadioButton rdbtnDefaultValues = new JRadioButton("Default Values");
rdbtnDefaultValues.setBounds(841, 583, 149, 23);
rdbtnDefaultValues.setBackground(Color.lightGray);
panel_1.add(rdbtnDefaultValues);

JLabel marc_prova2 = new JLabel("");
marc_prova2.setOpaque(true);
marc_prova2.setForeground(Color.LIGHT_GRAY);
marc_prova2.setBounds(61, 40, 275, 52);
panel_1.add(marc_prova2);

JLabel marc_prova3 = new JLabel("");
marc_prova3.setOpaque(true);
marc_prova3.setForeground(Color.LIGHT_GRAY);
marc_prova3.setBounds(805, 31, 535, 656);
panel_1.add(marc_prova3);
JLabel label = new JLabel("New label");
label.setBounds(0, 0, 1362, 720);
Image fondo = new ImageIcon(this.getClass().getResource("/fondo.jpg")).getImage();
label.setIcon(new ImageIcon(fondo));

JLabel label1 = new JLabel("New label");
label1.setBounds(0, 0, 1362, 720);
Image fondo1 = new ImageIcon(this.getClass().getResource("/fondo.jpg")).getImage();
label1.setIcon(new ImageIcon(fondo1));

panel.add(label);
panel_1.add(label1);
package com.northbound.settings;

public class Settings {

    // Chose the Opendaylight release control (HIDROGEN or HELIUM_or_LITHIUM)
    public static final String RELEASE = "HELIX";
    public static final String RELEASE = "HELIX_or_LITHIUM";

    // REST Parameters to log.
    public static final String USERNAME = "admin";
    public static final String PASSWORD = "admin";
    public static final String URL = choseURL(RELEASE);

    // IP address where the DPI is listening.
    public static final String HOST_DPI_LISTENING = "10.0.0.5";

    // Socket to connect with the DPI.
    public static final int SOCKET_PORT = 5000;

    // Option to Make our own Array to check the flows already installed, or check them
    // asking the controller the already installed flows.
    // If 0, we are creating our own Array
    // If 1, we are using REST API to ask the controller if it has the flow.
    public static int RESTReceiverOn = 0;

    public static final String SWITCH_TABLE = "0";  //Used to install the flows in
    //an specific table in Lithium

    public static final String HOST_MASK = "/32";  //Used to install the flows in
    //Lithium

    // Parameters for HELIX & LITHIUM Release

    // URL to install the flows
    public static final String FLOW_PROGRAMMER_REST_API = "/restconf/config/opendaylight-inventory:nodes/node/
    // IP addresses of the HOSTs.
    public static final String IP_HOST_1 = "10.0.0.1/32";
    public static final String IP_HOST_2 = "10.0.0.2/32";
public static final String IP_HOST_3 = "10.0.0.3/32";
public static final String IP_HOST_4 = "10.0.0.4/32";

// Node identifiers.
public static final String SWITCH_1 = "openflow:1";
public static final String SWITCH_2 = "openflow:2";
public static final String SWITCH_3 = "openflow:3";

// Input Ports per Switch
public static final String S1_in_PORT_1 = "openflow:1:1";
public static final String S1_in_PORT_2 = "openflow:1:2";
public static final String S1_in_PORT_3 = "openflow:1:3";
public static final String S2_in_PORT_1 = "openflow:2:1";
public static final String S2_in_PORT_2 = "openflow:2:2";
public static final String S2_in_PORT_4 = "openflow:2:4";
public static final String S3_in_PORT_1 = "openflow:3:1";

// Parameters for HIDROGEN Release

// URL to install the flows
public static final String FLOW_PROGRAMMER_REST_API = "/controller/nb/v2/flowprogrammer/default/node/OF/";

// IP addresses of the HOSTs.
public static final String IP_HOST_1 = "10.0.0.1";
public static final String IP_HOST_2 = "10.0.0.2";
public static final String IP_HOST_3 = "10.0.0.3";
public static final String IP_HOST_4 = "10.0.0.4";

// Node identifiers.
public static final String SWITCH_1 = "00:00:00:00:00:00:00:01";
public static final String SWITCH_2 = "00:00:00:00:00:00:00:02";
public static final String SWITCH_3 = "00:00:00:00:00:00:00:03";

// Input Ports per Switch
public static final String S1_in_PORT_1 = "1";
public static final String S1_in_PORT_2 = "2";
public static final String S1_in_PORT_3 = "3";
public static final String S2_in_PORT_1 = "1";
public static final String S2_in_PORT_2 = "2";
public static final String S2_in_PORT_4 = "4";
public static final String S3_in_PORT_1 = "1";
private static String choseURL(String RELEASE){
    String URL = null;
    if (RELEASE=="HIDROGEN"){
        URL = "http://192.168.56.101:8080";
    } else if (RELEASE=="HELIUM_or_LITHIUM"){
        URL = "http://192.168.56.101:8181";
    }
    return URL;
}
}
### A.11 External Orchestrator

```java
package com.northbound.mainflowinstaller;

import org.codehaus.jettison.json.JSONException;
import com.northbound.chainreader.ChainReader;
import com.northbound.flowconstructor.FlowInicializer;
import com.northbound.flowconstructor.FlowModel;
import com.northbound.flowconstructor.RestInterfaceJSONCreator;
import com.northbound.flowreceiver.RestInterfaceJSONAnalyserKaraf;
import com.northbound.flowreceiver.RestInterfaceJSONAnalyserHydrogen;
import com.northbound.settings.Settings;
import com.northbound.socket.StockQuoteServer;

public class ExternalOrchestrator {

    // public static void main(String[] args) throws JSONException {

    public static void FullAplication() throws JSONException {

        // Creation of a new Object (Model) to save each time the name, the @IPs and
        //the @IPd of the packet that we have obtained the chain by the DPI.
        FlowModel flowIn = new FlowModel();

        // Creation of a array of these Objects just to save every new flow and not
        //install flows already installed.
        FlowModel flowInArray[] = new FlowModel[55];

        // Initialization of the flows in the controller (ARP & IP). This flows are just
        //active until the DPI sent the information required to update them.
        FlowInicializer initFlow = new FlowInicializer();
        initFlow.installFlowInicializer();

        // loop to listen continuously the socket connected with the DPI and actualize
        //the flows.
        while(true){
            // Open the socket and receive the answers
            StockQuoteServer serverSocket = new StockQuoteServer();
            String flowSymbol = serverSocket.socketServer();
            System.out.println(flowSymbol);

            // Call to the function ChainReader to decrypt the chain received.
            ChainReader flowDecoded = new ChainReader(flowSymbol);
            int OUTPUT_PORTS[] = flowDecoded.getFlowDecoded();
            String actionToDoS1_go;
            String actionToDoS2_go;
```
String actionToDoS3_go;
String actionToDoS1_return = null;
String actionToDoS2_return = null;
String actionToDoS3_return = null;

// Getting the action to do.
if (OUTPUT_PORTS[0] != -1){  // If it is not a ICMPV6 or MDNS packet, that
  // do not have @IPs and @IPd.
    if (OUTPUT_PORTS[0] != -2){ // If the packet comes from a known source.
      actionToDoS1_go = Integer.toString(OUTPUT_PORTS[0]);
      actionToDoS2_go = Integer.toString(OUTPUT_PORTS[1]);
      actionToDoS3_go = Integer.toString(OUTPUT_PORTS[2]);

      actionToDoS1_return = Integer.toString(OUTPUT_PORTS[3]);
      actionToDoS2_return = Integer.toString(OUTPUT_PORTS[4]);
      actionToDoS3_return = Integer.toString(OUTPUT_PORTS[5]);
    } else { // If the packet comes from an unknown source.
      actionToDoS1_go = "DROP";
      actionToDoS2_go = "DROP";
      actionToDoS3_go = "DROP";
    }
}

// Getting the name, the @IPs and the @IPd of the packet that we have obtained
// the chain by the DPI.
String nwSrc_go = flowDecoded.getNwSrc();
String nwDst_go = flowDecoded.getNwDst();
String name_go = flowDecoded.getName() + ":from:" + nwSrc_go + ":dest:" + nwDst_go;

// In the come back way, the Source is interchanged with the destination.
String nwSrc_return = flowDecoded.getNwDst();
String nwDst_return = flowDecoded.getNwSrc();
String name_return = flowDecoded.getName() + ":from:" + nwDst_go + ":dest:" + nwSrc_go;

// Filling the Object Model to compare it with the previous Flows installed in
// the case that we are not using the
// REST interface to know the already
flowIn.setName(name_go);
flowIn.setNwSrc(nwSrc_go);
flowIn.setNwDst(nwDst_go);

if (Settings.RESTReciverOn == 1){
  if (Settings.RELEASE == "HIDROGEN"){
    RestInterfaceJSONAnalyserHydrogen recivingFlowsAlreadyInstalled =
      new RestInterfaceJSONAnalyserHydrogen();
flowInArray = receivingFlowsAlreadyInstalled.flowsAlreadyInstalled();
} else if (Settings.RELEASE == "HELIUM_or_LITHIUM"){
    RestInterfaceJSONAnalyserKaraf recivingFlowsAlreadyInstalled =
        new RestInterfaceJSONAnalyserKaraf();
    flowInArray = recivingFlowsAlreadyInstalled.flowsAlreadyInstalled();
}

// Installing the new flow.
int etherType = 2048;
int priority = 999;
String inPort = "-1";
String portNotUsed = "-1";
String portDPIListening = "5";
int j=0;
while (j < flowInArray.length){
    if (flowInArray[j] == null){  // If the flow have not been installed
        // before, we installed it now.
        RestInterfaceJSONCreator newInstall = new RestInterfaceJSONCreator();
        newInstall.flowInstallDPI(actionToDoS1_go, portDPIListening,
                          portNotUsed, priority, name_go, etherType, inPort, nwSrc_go,
                          nwDst_go, Settings.SWITCH_1);
        newInstall.flowInstallDPI(actionToDoS2_go, portNotUsed, portNotUsed,
                          priority, name_go, etherType, inPort, nwSrc_go, nwDst_go,
                          Settings.SWITCH_2);
        newInstall.flowInstallDPI(actionToDoS3_go, portNotUsed, portNotUsed,
                          priority, name_go, etherType, inPort, nwSrc_go, nwDst_go,
                          Settings.SWITCH_3);

        // Flows of the come back journey.
        if(actionToDoS1_return != null){
            newInstall.flowInstallDPI(actionToDoS1_return, portNotUsed,
                          portNotUsed, priority, name_return, etherType, inPort,
                          nwSrc_return, nwDst_return, Settings.SWITCH_1);
            newInstall.flowInstallDPI(actionToDoS2_return, portNotUsed, portNotUsed,
                          priority, name_return, etherType, inPort, nwSrc_return, nwDst_return,
                          Settings.SWITCH_2);
            newInstall.flowInstallDPI(actionToDoS3_return, portNotUsed, portNotUsed,
                          priority, name_return, etherType, inPort, nwSrc_return, nwDst_return,
                          Settings.SWITCH_3);
        }
    } // Filling the Comparing array if we are not using the REST
    // Receiving application.
    if (Settings.RESTReciverOn == 0){
        flowInArray[j] = new FlowModel();
        flowInArray[j].setName(name_go);
flowInArray[j].setNwSrc(nwSrc_go);
flowInArray[j].setNwDst(nwDst_go);
}

break;

}   // If the flow have been installed before,
   // we do not install it and we wait for the
   // next flow.
else if (flowIn.getName().equals(flowInArray[j].getName()) &&
 flowIn.getNwDst().equals(flowInArray[j].getNwDst()) &&
 flowIn.getNwSrc().equals(flowInArray[j].getNwSrc())){
    System.out.println("Same flow already installed previously");
    break;
}

}  else{
    j++;
}

} else{  // Omitting the ICMPV6 & MDNS packets that do not have @IPs
     // and @IPd.
    System.out.println("We omit this packet because it is a ICMPV6 or"
    "MDNS packet, that do not have @IPs and @IPd");
}

}
A.12 Flow Initializer

```java
package com.northbound.flowconstructor;

import org.codehaus.jettison.json.JSONException;
import com.northbound.settings.Settings;
import com.northbound.gui.GUIUserInterface;

public class FlowInitializer {
    public void installFlowInitializer() throws JSONException {
        // ARP
        // Data in common
        int priority = 501;
        int etherType = -1;
        RestInterfaceJSONCreator newFlowIP = new RestInterfaceJSONCreator();

        // OF13|1 - First Switch
        String nodeToInstall = Settings.SWITCH_1;
        newFlowIP.flowInstallDPI("2", "4", "5", priority, "S1:P1:arpFlowInsert", etherType, Settings.S1_in_PORT_1, "-1", "-1", nodeToInstall);

        // Port 2 listening: Variables between flows
        newFlowIP.flowInstallDPI("1", "4", "5", priority, "S1:P2:arpFlowInsert", etherType, Settings.S1_in_PORT_2, "-1", "-1", nodeToInstall);

        // Port 3 listening: Variables between flows
        newFlowIP.flowInstallDPI("1", "2", "4", "5", priority, "S1:P3:arpFlowInsert", etherType, Settings.S1_in_PORT_3, "-1", "-1", nodeToInstall);
    }
}
```
etherType, Settings.S1_in_PORT_3, "-1", "-1", nodeToInstall);

// OF13|2 - Second Switch --------------------------------------------------------
nodeToInstall = Settings.SWITCH_2;

// Port 1 listening: Variables between flows
// name: ("S2:P1:arpFlowInsert");
// in Port: ("1");
// out Port: ("2,3");
// Actual flow install:
etherType, Settings.S2_in_PORT_1, "-1", "-1", nodeToInstall);

// Port 2 listening: Variables between flows
// name: ("S2:P2:arpFlowInsert");
// in Port: ("2");
// out Port: ("1,3");
// Actual flow install:
etherType, Settings.S2_in_PORT_2, "-1", "-1", nodeToInstall);

// Port 4 listening: Variables between flows
// name: ("S2:P4:arpFlowInsert");
// in Port: ("4");
// out Port: ("1,2,3");
// Actual flow install:
etherType, Settings.S2_in_PORT_4, "-1", "-1", nodeToInstall);

// OF13|3 - Third Switch ----------------------------------------------------------
nodeToInstall = Settings.SWITCH_3;

// Port 1 listening: Variables between flows
// name: ("S3:P1:arpFlowInsert");
// in Port: ("1");
// out Port: ("2");
// Actual flow install:
etherType, Settings.S3_in_PORT_1, "-1", "-1", nodeToInstall);
A Code Developed

// IP
// Data in common
priority = 601;
etherType = 2048;
String inPort = "-1";
String IP_HOST_1 = Settings.IP_HOST_1;
String IP_HOST_2 = Settings.IP_HOST_2;
String IP_HOST_3 = Settings.IP_HOST_3;
String IP_HOST_4 = Settings.IP_HOST_4;

// OF13|1 - First Switch
nodeToInstall = Settings.SWITCH_1;

// H1 to H2: Variables between flows
// name: ("S1:H1-H2:InicialFlowInsert");
// IP_src: ("10.0.0.1");
// IP_dst: ("10.0.0.2");
// out Port: ("2");
// Actual flow install:
etherType, inPort, IP_HOST_1, IP_HOST_2, nodeToInstall);

// H1 to H3: Variables between flows
// name: ("S1:H1-H3:InicialFlowInsert");
// IP_src: ("10.0.0.1");
// IP_dst: ("10.0.0.3");
// out Port: ("=3,5");
// Actual flow install:
etherType, inPort, IP_HOST_1, IP_HOST_3, nodeToInstall);

// H1 to H4: Variables between flows
// name: ("S1:H1-H4:InicialFlowInsert");
// IP_src: ("10.0.0.1");
// IP_dst: ("10.0.0.4");
// Actual flow install:
etherType, inPort, IP_HOST_1, IP_HOST_4, nodeToInstall);

// From H2 to H1: Variables between flows
// name: ("S1:H2-H1:InicialFlowInsert");
// IP_src: ("10.0.0.2");
// IP_dst: ("10.0.0.1");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", ",", ",", priority, "S1:H2-H1:InicialFlowInsert", etherType, inPort, IP_HOST_2, IP_HOST_1, nodeToInstall);

// From H2 to H3: Variables between flows
// name: ("S1:H2-H3:InicialFlowInsert");
// IP_src: ("10.0.0.2");
// IP_dst: ("10.0.0.3");
// out Port: ("3,5");
// Actual flow install:
newFlowIP.flowInstallDPI("3", ",", ",", priority, "S1:H2-H3:InicialFlowInsert", etherType, inPort, IP_HOST_2, IP_HOST_3, nodeToInstall);

// From H2 to H4: Variables between flows
// name: ("S1:H2-H4:InicialFlowInsert");
// IP_src: ("10.0.0.2");
// IP_dst: ("10.0.0.4");
// Actual flow install:
newFlowIP.flowInstallDPI("3", ",", ",", priority, "S1:H2-H4:InicialFlowInsert", etherType, inPort, IP_HOST_2, IP_HOST_3, nodeToInstall);

// From H3 to H1: Variables between flows
// name: ("S1:H3-H1:InicialFlowInsert");
// IP_src: ("10.0.0.3");
// IP_dst: ("10.0.0.1");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", ",", ",", priority, "S1:H3-H1:InicialFlowInsert", etherType, inPort, IP_HOST_3, IP_HOST_1, nodeToInstall);

// From H3 to H2: Variables between flows
// name: ("S1:H3-H2:InicialFlowInsert");
// IP_src: ("10.0.0.3");
// IP_dst: ("10.0.0.2");
// out Port: ("2");
// Actual flow install:
newFlowIP.flowInstallDPI("2", ",", ",", priority, "S1:H3-H2:InicialFlowInsert", etherType, inPort, IP_HOST_3, IP_HOST_2, nodeToInstall);

// From H4 to H1: Variables between flows
// name: ("S1:H4-H1:InicialFlowInsert");
// IP_src: ("10.0.0.4");
// IP_dst: ("10.0.0.1");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", "-1", "-1", priority, "S1:H4-H1:InicialFlowInsert", etherType, inPort, IP_HOST_4, IP_HOST_1, nodeToInstall);

// From H4 to H2: Variables between flows
// name: ("S1:H4-H2:InicialFlowInsert");
// IP_src: ("10.0.0.4");
// IP_dst: ("10.0.0.2");
// out Port: ("2");
// Actual flow install:
newFlowIP.flowInstallDPI("2", "-1", "-1", priority, "S1:H4-H2:InicialFlowInsert", etherType, inPort, IP_HOST_4, IP_HOST_2, nodeToInstall);

// OF13|2 - Second Switch ---------------------------------------------------------------
nodeToInstall = Settings.SWITCH_2;

// FROM H1
// H1 to H3: Variables between flows
// name: ("S2:H1-H3:InicialFlowInsert");
// IP_src: ("10.0.0.1");
// IP_dst: ("10.0.0.3");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", "-1", "-1", priority, "S2:H1-H3:InicialFlowInsert", etherType, inPort, IP_HOST_1, IP_HOST_3, nodeToInstall);

// H1 to H4: Variables between flows
// name: ("S2:H1-H4:InicialFlowInsert");
// IP_src: ("10.0.0.1");
// IP_dst: ("10.0.0.4");
// out Port: ("2");
// Actual flow install:
newFlowIP.flowInstallDPI("2", "-1", "-1", priority, "S2:H1-H4:InicialFlowInsert", etherType, inPort, IP_HOST_1, IP_HOST_4, nodeToInstall);

// FROM H2
// H2 to H3: Variables between flows
// name: ("S2:H2-H3:InicialFlowInsert");
// IP_src: ("10.0.0.2");
// IP_dst: ("10.0.0.3");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", "-1", "-1", priority, "S2:H2-H3:InicialFlowInsert", etherType, inPort, IP_HOST_2, IP_HOST_3, nodeToInstall);
// H2 to H4: Variables between flows
// name:  ("S2:H2-H4:InicialFlowInsert");
// IP_src:  ("10.0.0.2");
// IP_dst:  ("10.0.0.4");
// out Port:  ("2");
// Actual flow install:
 etherType, inPort, IP_HOST_2, IP_HOST_4, nodeToInstall);

// FROM H3
// H3 to H1: Variables between flows
// name:  ("S2:H3-H1:InicialFlowInsert");
// IP_src:  ("10.0.0.3");
// IP_dst:  ("10.0.0.1");
// out Port:  ("3");
// Actual flow install:
 etherType, inPort, IP_HOST_3, IP_HOST_1, nodeToInstall);

// H3 to H2: Variables between flows
// name:  ("S2:H3-H2:InicialFlowInsert");
// IP_src:  ("10.0.0.3");
// IP_dst:  ("10.0.0.2");
// Actual flow install:
 etherType, inPort, IP_HOST_3, IP_HOST_2, nodeToInstall);

// H3 to H4: Variables between flows
// name:  ("S2:H3-H4:InicialFlowInsert");
// IP_src:  ("10.0.0.3");
// IP_dst:  ("10.0.0.4");
// out Port:  ("2");
// Actual flow install:
 etherType, inPort, IP_HOST_3, IP_HOST_4, nodeToInstall);

// FROM H4
// H4 to H1: Variables between flows
// name:  ("S2:H4-H1:InicialFlowInsert");
// IP_src:  ("10.0.0.4");
// IP_dst:  ("10.0.0.1");
// out Port:  ("3");
// Actual flow install:
newFlowIP.flowInstallDPI("3", ",", ",", priority, ",S2:H4-H1:InicialFlowInsert", etherType, inPort, IP_HOST_4, IP_HOST_1, nodeToInstall);

// H4 to H2: Variables between flows
// name: ("S2:H4-H2:InicialFlowInsert");
// IP_src: ("10.0.0.4");
// IP_dst: ("10.0.0.2");
// Actual flow install:
newFlowIP.flowInstallDPI("3", ",", ",", priority, ",S2:H4-H2:InicialFlowInsert", etherType, inPort, IP_HOST_4, IP_HOST_2, nodeToInstall);

// // H4 to H3: Variables between flows
// name: ("S2:H4-H3:InicialFlowInsert");
// IP_src: ("10.0.0.4");
// IP_dst: ("10.0.0.3");
// out Port: ("1");
// Actual flow install:
newFlowIP.flowInstallDPI("1", ",", ",", priority, ",S2:H4-H3:InicialFlowInsert", etherType, inPort, IP_HOST_4, IP_HOST_3, nodeToInstall);
package com.northbound.flowconstructor;

import org.codehaus.jettison.json.JSONArray;
import org.codehaus.jettison.json.JSONException;
import org.codehaus.jettison.json.JSONObject;

import com.northbound.flowsender.RestInterfaceSender;
import com.northbound.settings.Settings;

public class RestInterfaceJSONCreator {

    public void flowInstallDPI(String actionToDo1, String actionToDo2,
                           String actionToDo3, int priority, String name, int etherType,
                           String inPort, String nwSrc, String nwDst, String nodeToInstall)
    throws JSONException{

        if (Settings.RELEASE=="HELIUM_or_LITHIUM"){

            // Table in the REST API where we want to install the Flow
            String table = Settings.SWITCH_TABLE;

            // Translation of IPs. In Helium & in Lithium the Mask is needed.
            nwSrc = IPsrcVerifier(nwSrc);
            nwDst = IPdstVerifier(nwDst);

            JSONObject outputAction1 = new JSONObject();
            outputAction1.put("output-node-connector", actionToDo1);

            JSONObject insideAction1 = new JSONObject();
            insideAction1.put("order", 0);
            insideAction1.put("output-action", outputAction1);
            JSONArray action = new JSONArray();
            action.put(insideAction1);

            if (!actionToDo2.equals("-1")){
                JSONObject outputAction2 = new JSONObject();
                outputAction2.put("output-node-connector", actionToDo2);

                JSONObject insideAction2 = new JSONObject();
                insideAction2.put("order", 1);
                insideAction2.put("output-action", outputAction2);
                action.put(insideAction2);
            }
        }
    }
}
if (!actionToDo3.equals("-1")){
    JSONObject outputAction3 = new JSONObject();
    outputAction3.put("output-node-connector", actionToDo3);

    JSONObject insideAction3 = new JSONObject();
    insideAction3.put("order", 2);
    insideAction3.put("output-action", outputAction3);
    action.put(insideAction3);
}

JSONObject applyActions = new JSONObject();
applyActions.put("action", action);

JSONObject insideInstruction = new JSONObject();
insideInstruction.put("order", 0);
insideInstruction.put("apply-actions", applyActions);

JSONArray insideInstructionArray = new JSONArray();
insideInstructionArray.put(insideInstruction);

JSONObject Instruction = new JSONObject();
Instruction.put("instruction", insideInstructionArray);

JSONObject flowInside = new JSONObject();
flowInside.put("id", name);
flowInside.put("instructions", Instruction);
flowInside.put("priority", priority);
flowInside.put("flow-name", name);

JSONObject ethernetMatch = new JSONObject();
// ARP Packet FLOW
if(etherType!=-1){
    JSONObject ethernetType = new JSONObject();
    ethernetType.put("type", etherType);
    ethernetMatch.put("ethernet-type", ethernetType);
}

JSONObject match = new JSONObject();
if(nwSrc!="-1"){
    match.put("ipv4-destination", nwDst);
    match.put("ipv4-source", nwSrc);
} else{
    match.put("in-port", inPort);
}
if(etherType!=-1){
    match.put("ethernet-match", ethernetMatch);
}
flowInside.put("match", match);
flowInside.put("table_id", 0);
flowInside.put("priority", priority);

JSONArray flowInsideArray = new JSONArray();
flowInsideArray.put(flowInside);

JSONObject flow = new JSONObject();
flow.put("flow", flowInsideArray);

// Setting the URL where to install the flow to call it
String FLOW_PROGRAMMER_REST_API = Settings.FLOW_PROGRAMMER_REST_API;
String baseURL = Settings.URL + FLOW_PROGRAMMER_REST_API + nodeToInstall + "/table/" + table + "/flow/" + name;

// Actual flow install
RestInterfaceSender.installFlow(baseURL, flow);

// HIDROGEN RELEASE
else if(Settings.RELEASE=="HIDROGEN"){
    String actionToDo;
    if(actionToDo2.equals("-1") || actionToDo2.equals("DROP") ){
        actionToDo = "OUTPUT=" + actionToDo1;
    } else{
        if(actionToDo3.equals("-1"){
            actionToDo = "OUTPUT=" + actionToDo1 + "," + actionToDo2;
        } else{
            actionToDo = "OUTPUT=" + actionToDo1 + "," + actionToDo2 + "," + actionToDo3;
        }
    }

    // Flow Parameters and Actions to install the new flow that will supply the
    // flow installed by the Inicializator.
    JSONObject flow = new JSONObject();
    flow.put("name", name);
    if(!nwSrc.equals("-1")){
        flow.put("nwSrc", nwSrc);
    flow.put("nwDst", nwDst);
} else{
    flow.put("ingressPort", inPort);
}

// flow.put("installInHw", "true");
flow.put("priority", priority);
if(etherType!=-1){
    flow.put("etherType", etherType);
}
flow.put("actions", new JSONArray().put(actionToDo));

// Node on which this flow will be installed
JSONObject node = new JSONObject();
node.put("id", nodeToInstall);
node.put("type", "OF");
flow.put("node", node);

// Setting the URL where to install the flow to call it
String FLOW_PROGRAMMER_REST_API = Settings.FLOW_PROGRAMMER_REST_API;
String baseURL = Settings.URL + FLOW_PROGRAMMER_REST_API + nodeToInstall + "/staticFlow/" + name;

// Installation of the flow.
RestInterfaceSender.installFlow(baseURL, flow);
}

// FUNCTIONS

private String IPsrcVerifier(String nwSrc){
    if (nwSrc.equals("10.0.0.1")){
        nwSrc = nwSrc + Settings.HOST_MASK;
    }
    if (nwSrc.equals("10.0.0.2")){
        nwSrc = nwSrc + Settings.HOST_MASK;
    }
    if (nwSrc.equals("10.0.0.3")){
        nwSrc = nwSrc + Settings.HOST_MASK;
    }
    if (nwSrc.equals("10.0.0.4")){
        nwSrc = nwSrc + Settings.HOST_MASK;
return nwSrc;

private String IPdstVerifier(String nwDst){
    if (nwDst.equals("10.0.0.1")){
        nwDst = nwDst + Settings.HOST_MASK;
    }
    if (nwDst.equals("10.0.0.2")){
        nwDst = nwDst + Settings.HOST_MASK;
    }
    if (nwDst.equals("10.0.0.3")){
        nwDst = nwDst + Settings.HOST_MASK;
    }
    if (nwDst.equals("10.0.0.4")){
        nwDst = nwDst + Settings.HOST_MASK;
    }
    return nwDst;
}
A.14 Rest Interface Sender

```java
package com.northbound.flowsender;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import org.apache.commons.codec.binary.Base64;
import org.codehaus.jettison.json.JSONObject;
import com.northbound.settings.Settings;

public class RestInterfaceSender {

    private static String FLOW_PROGRAMMER_REST_API = Settings.FLOW_PROGRAMMER_REST_API;

    private static final int ACTUALIZED = 200;
    private static final int NO_CONTENT = 204;
    private static final int CREATED = 201;

    public static boolean installFlow(String baseURL, JSONObject flow) {
        HttpURLConnection connection = null;
        int callStatus = 0;

        try {
            URL url = new URL(baseURL);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("PUT");
            connection.setRequestProperty("Authorization", "Basic " + Base64.encodeBase64String(flow.getBytes()));
            connection.setRequestProperty("Content-Type", "application/json");
            connection.setUseCaches(false);

            callStatus = connection.getResponseCode();
            if (callStatus == ACTUALIZED) {
                return true;
            }

            return false;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}
```
connection.setDoInput(true);
connection.setDoOutput(true);

// Set Post Data
OutputStream os = connection.getOutputStream();

String flowToInstall = flow.toString();

os.write(flowToInstall.getBytes());

os.close();

// Getting the response code
callStatus = connection.getResponseCode();

} catch (Exception e) {
    System.err.println("Unexpected error while flow installation.. "
            + e.getMessage());
e.printStackTrace();
} finally {
    if (connection != null)
        connection.disconnect();
}

if (callStatus == CREATED) {
    System.out.println("Flow installed Successfully");
    return true;
} else {
    if(callStatus != ACTUALIZED){
        System.err.println("Failed to install flow.. " + callStatus);
    } else{
        System.err.println("Flow actualized.. " + callStatus);
    }
    return false;
}

public static boolean deleteFlow(String flowName, String nodeId) {
    HttpURLConnection connection = null;
    int callStatus = 0;
    String baseURL = Settings.URL + FLOW_PROGRAMMER_REST_API + nodeId + "/staticFlow/" + flowName;

    try {
        // Create URL = base URL + container
        URL url = new URL(baseURL);
// Create authentication string and encode it to Base64
String authStr = Settings.USERNAME + ":" + Settings.PASSWORD;
String encodedAuthStr = Base64.encodeBase64String(authStr.getBytes());

// Create Http connection
connection = (HttpURLConnection) url.openConnection();

// Set connection properties
connection.setRequestMethod("DELETE");
connection.setRequestProperty("Authorization", "Basic " + encodedAuthStr);
connection.setRequestProperty("Content-Type", "application/json");
callStatus = connection.getResponseCode();

} catch (Exception e) {
    System.err.println("Unexpected error while flow deletion."
        + e.getMessage());
e.printStackTrace();
} finally {
    if (connection != null)
        connection.disconnect();
}

if (callStatus == NO_CONTENT) {
    System.out.println("Flow deleted Successfully");
    return true;
} else {
    System.err.println("Failed to delete the flow...
        " + callStatus + \"n\"");
    return false;
}
}
public class StockQuoteServer {
    // Server Socket to get the analysis of the DPI(Socket Client)
    //
    public String socketServer() {
        ServerSocket serverSocket = null;
        Socket client = null;
        BufferedReader inbound = null;
        String symbolFlow = null;
        try {
            //Server Socket
            serverSocket = new ServerSocket(Settings.SOCKET_PORT);

            System.out.println("\nWaiting a request");

            //Waiting the client
            client = serverSocket.accept();

            //Obtaining the fluxes
            inbound = new BufferedReader(new InputStreamReader(client.getInputStream()));
            symbolFlow = inbound.readLine();
            // System.out.println(symbolFlow);
        } catch (IOException ioe) {
            System.out.println("Error in the server: "+ ioe);
        }
        finally {
            try{
                serverSocket.close();
                inbound.close();
            } catch(Exception e){
                System.out.println("Can’t close the stream " + e.getMessage());
            }
        }
        return symbolFlow;
    }
}
A.16 Chain Reader

```java
package com.northbound.chainreader;

public class ChainReader {
    private String flowSymbol; // Chain obtained from the DPI.
    private String[] arrayWords; // Chain separated by Strings.
    private int length; // Number of Strings that have the chain.

    // Constructor: receiving the chain.
    public ChainReader(String flowSymbol) {
        this.flowSymbol = flowSymbol;
    }

    // Method to return the ports to flow the packet depending on the:
    // Number of the Switch, @IPs, @IPd, Type of protocol.
    public int[] getFlowDecoded() {
        int OUTPUT_PORT_SWITCH_1_go = 0;
        int OUTPUT_PORT_SWITCH_2_go = 0;
        int OUTPUT_PORT_SWITCH_3_go = 0;

        int OUTPUT_PORT_SWITCH_1_return = 0;
        int OUTPUT_PORT_SWITCH_2_return = 0;
        int OUTPUT_PORT_SWITCH_3_return = 0;

        int IPNumSrc_go;
        int IPNumDst_go;

        int IPNumSrc_return;
        int IPNumDst_return;

        setArrayWords(flowSymbol);

        if (length > 7) {
            OUTPUT_PORT_SWITCH_1_go = -1; // ICMPv6 & MDNS packets that do not
            // have @IPs and @IPd.
        } else {
            FullMatrix matrix = new FullMatrix();
            IPNumSrc_go = decodeSourceIP(arrayWords[0]);
            IPNumDst_go = decodeDestinationIP(arrayWords[2]);

            // To come back, the Source and the Destinations are just interchanged.
        }
    }
}
```
```java
IPNumSrc_return = decodeDestinationIP(arrayWords[2]);
IPNumDst_return = decodeSourceIP(arrayWords[0]);

if (IPNumSrc_go != -1) {
    // The packet is from a known source.
    OUTPUT_PORT_SWITCH_1_go = matrix.getOutputPortSwitch1(IPNumSrc_go,
                Integer.parseInt(arrayWords[5]));
    OUTPUT_PORT_SWITCH_2_go = matrix.getOutputPortSwitch2(IPNumSrc_go,
                IPNumDst_go, Integer.parseInt(arrayWords[5]));
    OUTPUT_PORT_SWITCH_3_go = matrix.getOutputPortSwitch3(IPNumSrc_go,
                Integer.parseInt(arrayWords[5]), 0);

    // Using the symmetric property of the 4 firsts ports of our Switches,
    // we will use the same matrix to get the ports of the come back way
    // of the packet.
    OUTPUT_PORT_SWITCH_2_return = matrix.getOutputPortSwitch1(IPNumSrc_go,
                  Integer.parseInt(arrayWords[5]));
    OUTPUT_PORT_SWITCH_1_return = matrix.getOutputPortSwitch2(IPNumSrc_return,
                  IPNumDst_return, Integer.parseInt(arrayWords[5]));
    OUTPUT_PORT_SWITCH_3_return = matrix.getOutputPortSwitch3(IPNumSrc_return,
                  Integer.parseInt(arrayWords[5]), 1);
}
else {  // The packet is from an unknown source.
    OUTPUT_PORT_SWITCH_1_go = -2;
}

int[] OUTPUT_PORTS = new int[]{OUTPUT_PORT_SWITCH_1_go, OUTPUT_PORT_SWITCH_2_go,
                               OUTPUT_PORT_SWITCH_3_go, OUTPUT_PORT_SWITCH_1_return,
                               OUTPUT_PORT_SWITCH_2_return, OUTPUT_PORT_SWITCH_3_return};

return OUTPUT_PORTS;
}
```

// Method that separate the chain in the strings with 'usually' the next format-->  
private void setArrayWords(String flowSymbol){  
    arrayWords = flowSymbol.split(":");  
    lenght = arrayWords.length;  
}

// Method to decode the @IPs of the chain sent by the DPI  
public int decodeSourceIP(String IPs){  
    if (IPs.equals("10.0.0.1"))
```
return 0;
else if (IPs.equals("10.0.0.2"))
    return 1;
else
    return -1;
}

// Method to decode the @IPd of the chain sent by the DPI
public int decodeDestinationIP(String IPs){
    if (IPs.equals("10.0.0.3"))
        return 0;
    else if (IPs.equals("10.0.0.4"))
        return 1;
    else
        return -1;
}

// Getter of the @IPs of the chain sent by the DPI; Returns the @IPs
public String getNwSrc(){
    if (lenght > 7) {
        return "-1";
    } else{
        return arrayWords[0];
    }
}

// Getter of the @IPd of the chain sent by the DPI; Returns the @IPd
public String getNwDst(){
    if (lenght > 7) {
        return "-1";
    } else{
        return arrayWords[2];
    }
}

// Getter of the Name of the chain sent by the DPI; Returns the Name
public String getName(){
    return arrayWords[lenght-1];
}
package com.northbound.chainreader;

public class FullMatrix {

    // Matrix to return the ports to flow the packet depending on the:
    // [Number of the Switch][@IPs][@IPd][Type of protocol].
    private int[][][][] matrix = new int[4][2][2][200];

    public FullMatrix(){
        // First Switch
        // Class A
        matrix[0][0][0][2] = 3; // POP3
        matrix[0][0][0][3] = 3; // SMTP
        matrix[0][0][0][4] = 3; // IMAP
        matrix[0][0][0][65] = 3; // IRC
        matrix[0][0][0][119] = 3; // Facebook
        matrix[0][0][0][120] = 3; // Twitter
        matrix[0][0][0][122] = 3; // GMail
        matrix[0][0][0][186] = 3; // FacebookChat
        matrix[0][0][0][5] = 3; // DNS
        matrix[0][0][0][14] = 3; // SNMP
        matrix[0][0][0][17] = 3; // Syslog
        matrix[0][0][0][18] = 3; // DHCP
        matrix[0][0][0][81] = 3; // ICMP
        matrix[0][0][0][112] = 3; // LDAP
        matrix[0][0][0][7] = 3; // HTTP
        matrix[0][0][0][70] = 3; // Yahoo
        matrix[0][0][0][123] = 3; // GoogleMaps
        matrix[0][0][0][126] = 3; // Google
        matrix[0][0][0][178] = 3; // Amazon
        matrix[0][0][0][179] = 3; // eBay
        matrix[0][0][0][180] = 3; // CNN
        matrix[0][0][0][37] = 3; // BitTorrent
        matrix[0][0][0][39] = 3; // AVI
        matrix[0][0][0][40] = 3; // Flash
        matrix[0][0][0][43] = 3; // QuickTime
        matrix[0][0][0][45] = 3; // WindowsMedia
        matrix[0][0][0][100] = 3; // SIP
        matrix[0][0][0][121] = 3; // DropBox
        matrix[0][0][0][124] = 3; // Youtube
        matrix[0][0][0][125] = 3; // Skype
        matrix[0][0][0][92] = 3; // SSH
        matrix[0][0][0][159] = 3; // OpenVPN
        matrix[0][0][0][163] = 3; // TOR
// Class B
matrix[0][1][0][2] = 4;  // POP3
matrix[0][1][0][3] = 4;  // SMTP
matrix[0][1][0][4] = 4;  // IMAP
matrix[0][1][0][65] = 4; // IRC
matrix[0][1][0][119] = 4; // Facebook
matrix[0][1][0][120] = 4; // Twitter
matrix[0][1][0][122] = 4; // GMail
matrix[0][1][0][186] = 4; // FacebookChat
matrix[0][1][0][5] = 4;  // DNS
matrix[0][1][0][14] = 4; // SNMP
matrix[0][1][0][17] = 4; // Syslog
matrix[0][1][0][18] = 4; // DHCP
matrix[0][1][0][81] = 4; // ICMP
matrix[0][1][0][7] = 4;  // HTTP
matrix[0][1][0][70] = 4; // Yahoo
matrix[0][1][0][123] = 4; // GoogleMaps
matrix[0][1][0][126] = 4; // Google
matrix[0][1][0][178] = 4; // Amazon
matrix[0][1][0][179] = 4; // eBay
matrix[0][1][0][180] = 4; // CNN
matrix[0][1][0][37] = 4; // BitTorrent
matrix[0][1][0][39] = 4; // AVI
matrix[0][1][0][40] = 4; // Flash
matrix[0][1][0][43] = 4; // QuickTime
matrix[0][1][0][45] = 4; // WindowsMedia
matrix[0][1][0][100] = 4; // SIP
matrix[0][1][0][121] = 4; // DropBox
matrix[0][1][0][124] = 4; // Youtube
matrix[0][1][0][125] = 4; // Skype
matrix[0][1][0][92] = 4; // SSH
matrix[0][1][0][159] = 4; // OpenVPN
matrix[0][1][0][163] = 4; // TOR

// Second Switch --H3-- destination

// Class A
matrix[1][0][0][2] = 1;  // POP3
matrix[1][0][0][3] = 1;  // SMTP
matrix[1][0][0][4] = 1;  // IMAP
matrix[1][0][0][65] = 1; // IRC
matrix[1][0][0][119] = 1; // Facebook
matrix[1][0][0][120] = 1; // Twitter
matrix[1][0][0][122] = 1; // GMail
matrix[1][0][0][186] = 1;  // FacebookChat
matrix[1][0][0][6] = 1;  // DNS
matrix[1][0][0][14] = 1;  // SNMP
matrix[1][0][0][17] = 1;  // Syslog
matrix[1][0][0][18] = 1;  // DHCP
matrix[1][0][0][81] = 1;  // ICMP
matrix[1][0][0][112] = 1;  // LDAP
matrix[1][0][0][7] = 1;  // HTTP
matrix[1][0][0][70] = 1;  // Yahoo
matrix[1][0][0][123] = 1;  // GoogleMaps
matrix[1][0][0][126] = 1;  // Google
matrix[1][0][0][178] = 1;  // Amazon
matrix[1][0][0][179] = 1;  // eBay
matrix[1][0][0][180] = 1;  // CNN
matrix[1][0][0][37] = 1;  // BitTorrent
matrix[1][0][0][39] = 1;  // AVI
matrix[1][0][0][40] = 1;  // Flash
matrix[1][0][0][43] = 1;  // QuickTime
matrix[1][0][0][45] = 1;  // WindowsMedia
matrix[1][0][0][100] = 1;  // SIP
matrix[1][0][0][121] = 1;  // DropBox
matrix[1][0][0][124] = 1;  // Youtube
matrix[1][0][0][125] = 1;  // Skype
matrix[1][0][0][92] = 1;  // SSH
matrix[1][0][0][159] = 1;  // OpenVPN
matrix[1][0][0][163] = 1;  // TOR

// Class B
matrix[1][1][0][2] = 1;  // POP3
matrix[1][1][0][3] = 1;  // SMTP
matrix[1][1][0][4] = 1;  // IMAP
matrix[1][1][0][65] = 1;  // IRC
matrix[1][1][0][119] = 1;  // Facebook
matrix[1][1][0][120] = 1;  // Twitter
matrix[1][1][0][122] = 1;  // GMail
matrix[1][1][0][186] = 1;  // FacebookChat
matrix[1][1][0][6] = 1;  // DNS
matrix[1][1][0][14] = 1;  // SNMP
matrix[1][1][0][17] = 1;  // Syslog
matrix[1][1][0][18] = 1;  // DHCP
matrix[1][1][0][81] = 1;  // ICMP
matrix[1][1][0][112] = 1;  // LDAP
matrix[1][1][0][7] = 1;  // HTTP
matrix[1][1][0][70] = 1;  // Yahoo
matrix[1][1][0][123] = 1;  // GoogleMaps
matrix[1][1][0][126] = 1;  // Google
matrix[1][1][0][178] = 1;  // Amazon
matrix[1][1][0][179] = 1;  // eBay
matrix[1][1][0][180] = 1;  // CNN
\[ \begin{align*}
\text{matrix}[1][1][0][37] &= 1; & \text{// BitTorrent} \\
\text{matrix}[1][1][0][39] &= 1; & \text{// AVI} \\
\text{matrix}[1][1][0][40] &= 1; & \text{// Flash} \\
\text{matrix}[1][1][0][43] &= 1; & \text{// QuickTime} \\
\text{matrix}[1][1][0][45] &= 1; & \text{// Windows\_W\_d\_ia} \\
\text{matrix}[1][1][0][100] &= 1; & \text{// SIP} \\
\text{matrix}[1][1][0][121] &= 1; & \text{// DropBox} \\
\text{matrix}[1][1][0][124] &= 1; & \text{// Youtube} \\
\text{matrix}[1][1][0][125] &= 1; & \text{// Skype} \\
\text{matrix}[1][1][0][92] &= 1; & \text{// SSH} \\
\text{matrix}[1][1][0][159] &= 1; & \text{// Open\_VPN} \\
\text{matrix}[1][1][0][163] &= 1; & \text{// TOR} \\
\end{align*} \]

// Second Switch --H4-- destination

// Class A
\[ \begin{align*}
\text{matrix}[1][0][1][2] &= 2; & \text{// POP3} \\
\text{matrix}[1][0][1][3] &= 2; & \text{// SMTP} \\
\text{matrix}[1][0][1][4] &= 2; & \text{// IMAP} \\
\text{matrix}[1][0][1][65] &= 2; & \text{// IRC} \\
\text{matrix}[1][0][1][119] &= 2; & \text{// Facebook} \\
\text{matrix}[1][0][1][120] &= 2; & \text{// Twitter} \\
\text{matrix}[1][0][1][122] &= 2; & \text{// GMail} \\
\text{matrix}[1][0][1][186] &= 2; & \text{// Facebook\_Chat} \\
\text{matrix}[1][0][1][5] &= 2; & \text{// DNS} \\
\text{matrix}[1][0][1][14] &= 2; & \text{// SNMP} \\
\text{matrix}[1][0][1][17] &= 2; & \text{// Syslog} \\
\text{matrix}[1][0][1][18] &= 2; & \text{// DHCP} \\
\text{matrix}[1][0][1][81] &= 2; & \text{// ICMP} \\
\text{matrix}[1][0][1][112] &= 2; & \text{// LDAP} \\
\text{matrix}[1][0][1][7] &= 2; & \text{// HTTP} \\
\text{matrix}[1][0][1][70] &= 2; & \text{// Yahoo} \\
\text{matrix}[1][0][1][123] &= 2; & \text{// Google\_Maps} \\
\text{matrix}[1][0][1][126] &= 2; & \text{// Google} \\
\text{matrix}[1][0][1][178] &= 2; & \text{// Amazon} \\
\text{matrix}[1][0][1][179] &= 2; & \text{// eBay} \\
\text{matrix}[1][0][1][180] &= 2; & \text{// CNN} \\
\text{matrix}[1][0][1][37] &= 2; & \text{// BitTorrent} \\
\text{matrix}[1][0][1][39] &= 2; & \text{// AVI} \\
\text{matrix}[1][0][1][40] &= 2; & \text{// Flash} \\
\text{matrix}[1][0][1][43] &= 2; & \text{// QuickTime} \\
\text{matrix}[1][0][1][45] &= 2; & \text{// Windows\_W\_d\_ia} \\
\text{matrix}[1][0][1][100] &= 2; & \text{// SIP} \\
\text{matrix}[1][0][1][121] &= 2; & \text{// DropBox} \\
\text{matrix}[1][0][1][124] &= 2; & \text{// Youtube} \\
\text{matrix}[1][0][1][125] &= 2; & \text{// Skype} \\
\text{matrix}[1][0][1][92] &= 2; & \text{// SSH} \\
\end{align*} \]
matrix[1][0][1][159] = 2;   // OpenVPN
matrix[1][0][1][163] = 2;   // TOR

// Class B
matrix[1][1][1][2] = 2;      // POP3
matrix[1][1][1][3] = 2;      // SMTP
matrix[1][1][1][4] = 2;      // IMAP
matrix[1][1][1][65] = 2;     // IRC
matrix[1][1][1][119] = 2;    // Facebook
matrix[1][1][1][120] = 2;    // Twitter
matrix[1][1][1][122] = 2;    // GMail
matrix[1][1][1][186] = 2;    // FacebookChat
matrix[1][1][1][5] = 2;      // DNS
matrix[1][1][1][14] = 2;     // SNMP
matrix[1][1][1][17] = 2;     // Syslog
matrix[1][1][1][18] = 2;     // DHCP
matrix[1][1][1][81] = 2;     // ICMP
matrix[1][1][1][112] = 2;    // LDAP
matrix[1][1][1][7] = 2;      // HTTP
matrix[1][1][1][70] = 2;     // Yahoo
matrix[1][1][1][123] = 2;    // GoogleMaps
matrix[1][1][1][126] = 2;    // Google
matrix[1][1][1][178] = 2;    // Amazon
matrix[1][1][1][179] = 2;    // eBay
matrix[1][1][1][180] = 2;    // CNN
matrix[1][1][1][37] = 2;     // BitTorrent
matrix[1][1][1][39] = 2;     // AVI
matrix[1][1][1][40] = 2;     // Flash
matrix[1][1][1][43] = 2;     // QuickTime
matrix[1][1][1][45] = 2;     // WindowsMwdia
matrix[1][1][1][100] = 2;    // SIP
matrix[1][1][1][121] = 2;    // DropBox
matrix[1][1][1][124] = 2;    // Youtube
matrix[1][1][1][125] = 2;    // Skype
matrix[1][1][1][92] = 2;     // SSH
matrix[1][1][1][159] = 2;    // OpenVPN
matrix[1][1][1][163] = 2;    // TOR

//--------------------------------------------
// Third Switch
// Class A
matrix[2][0][0][2] = 2;    // POP3
matrix[2][0][0][3] = 2;    // SMTP
matrix[2][0][0][4] = 2;    // IMAP
matrix[2][0][0][65] = 2;   // IRC
matrix[2][0][0][119] = 2;  // Facebook
matrix[2][0][0][120] = 2;  // Twitter
matrix[2][0][0][122] = 2;  // GMail
matrix[2][0][0][186] = 2;  // FacebookChat
matrix[2][0][0][5] = 2;    // DNS
matrix[2][0][0][14] = 2;   // SNMP
matrix[2][0][0][17] = 2;   // Syslog
matrix[2][0][0][18] = 2;   // DHCP
matrix[2][0][0][81] = 2;   // ICMP
matrix[2][0][0][112] = 2;  // LDAP
matrix[2][0][0][7] = 2;    // HTTP
matrix[2][0][0][70] = 2;   // Yahoo
matrix[2][0][0][123] = 2;  // GoogleMaps
matrix[2][0][0][126] = 2;  // Google
matrix[2][0][0][178] = 2;  // Amazon
matrix[2][0][0][179] = 2;  // eBay
matrix[2][0][0][180] = 2;  // CNN
matrix[2][0][0][37] = 2;   // BitTorrent
matrix[2][0][0][39] = 2;   // AVI
matrix[2][0][0][40] = 2;   // Flash
matrix[2][0][0][43] = 2;   // QuickTime
matrix[2][0][0][45] = 2;   // WindowsMedia
matrix[2][0][0][100] = 2;  // SIP
matrix[2][0][0][121] = 2;  // DropBox
matrix[2][0][0][124] = 2;  // Youtube
matrix[2][0][0][125] = 2;  // Skype
matrix[2][0][0][92] = 2;   // SSH
matrix[2][0][0][159] = 2;  // OpenVPN
matrix[2][0][0][163] = 2;  // TOR

// Class B
matrix[2][1][0][2] = 2;    // POP3
matrix[2][1][0][3] = 2;    // SMTP
matrix[2][1][0][4] = 2;    // IMAP
matrix[2][1][0][65] = 2;   // IRC
matrix[2][1][0][119] = 2;  // Facebook
matrix[2][1][0][120] = 2;  // Twitter
matrix[2][1][0][122] = 2;  // GMail
matrix[2][1][0][186] = 2;  // FacebookChat
matrix[2][1][0][5] = 2;    // DNS
matrix[2][1][0][14] = 2;   // SNMP
matrix[2][1][0][17] = 2;   // Syslog
matrix[2][1][0][18] = 2;   // DHCP
matrix[2][1][0][81] = 2;   // ICMP
matrix[2][1][0][112] = 2;  // LDAP
\[\text{matrix}[2][1][0][7] = 2; \quad \text{// HTTP} \]
\[\text{matrix}[2][1][0][70] = 2; \quad \text{// Yahoo} \]
\[\text{matrix}[2][1][0][123] = 2; \quad \text{// GoogleMaps} \]
\[\text{matrix}[2][1][0][126] = 2; \quad \text{// Google} \]
\[\text{matrix}[2][1][0][178] = 2; \quad \text{// Amazon} \]
\[\text{matrix}[2][1][0][179] = 2; \quad \text{// eBay} \]
\[\text{matrix}[2][1][0][180] = 2; \quad \text{// CNN} \]
\[\text{matrix}[2][1][0][37] = 2; \quad \text{// BitTorrent} \]
\[\text{matrix}[2][1][0][39] = 2; \quad \text{// AVI} \]
\[\text{matrix}[2][1][0][40] = 2; \quad \text{// Flash} \]
\[\text{matrix}[2][1][0][43] = 2; \quad \text{// QuickTime} \]
\[\text{matrix}[2][1][0][45] = 2; \quad \text{// WindowsMedia} \]
\[\text{matrix}[2][1][0][100] = 2; \quad \text{// SIP} \]
\[\text{matrix}[2][1][0][121] = 2; \quad \text{// Dropbox} \]
\[\text{matrix}[2][1][0][124] = 2; \quad \text{// Youtube} \]
\[\text{matrix}[2][1][0][125] = 2; \quad \text{// Skype} \]
\[\text{matrix}[2][1][0][129] = 2; \quad \text{// CNN} \]
\[\text{matrix}[2][1][0][92] = 2; \quad \text{// SSH} \]
\[\text{matrix}[2][1][0][124] = 2; \quad \text{// OpenVPN} \]
\[\text{matrix}[2][1][0][163] = 2; \quad \text{// TOR} \]

//-------------------------------
// Third Switch Come Back way
// Class A
\[\text{matrix}[3][0][0][2] = 1; \quad \text{// POP3} \]
\[\text{matrix}[3][0][0][3] = 1; \quad \text{// SMTP} \]
\[\text{matrix}[3][0][0][4] = 1; \quad \text{// IMAP} \]
\[\text{matrix}[3][0][0][65] = 1; \quad \text{// IRC} \]
\[\text{matrix}[3][0][0][119] = 1; \quad \text{// Facebook} \]
\[\text{matrix}[3][0][0][120] = 1; \quad \text{// Twitter} \]
\[\text{matrix}[3][0][0][122] = 1; \quad \text{// GMail} \]
\[\text{matrix}[3][0][0][186] = 1; \quad \text{// FacebookChat} \]
\[\text{matrix}[3][0][0][5] = 1; \quad \text{// DNS} \]
\[\text{matrix}[3][0][0][14] = 1; \quad \text{// SNMP} \]
\[\text{matrix}[3][0][0][17] = 1; \quad \text{// Syslog} \]
\[\text{matrix}[3][0][0][18] = 1; \quad \text{// DHCP} \]
\[\text{matrix}[3][0][0][81] = 1; \quad \text{// ICMP} \]
\[\text{matrix}[3][0][0][112] = 1; \quad \text{// LDAP} \]
\[\text{matrix}[3][0][0][7] = 1; \quad \text{// HTTP} \]
\[\text{matrix}[3][0][0][70] = 1; \quad \text{// Yahoo} \]
\[\text{matrix}[3][0][0][123] = 1; \quad \text{// GoogleMaps} \]
\[\text{matrix}[3][0][0][126] = 1; \quad \text{// Google} \]
\[\text{matrix}[3][0][0][178] = 1; \quad \text{// Amazon} \]
\[\text{matrix}[3][0][0][179] = 1; \quad \text{// eBay} \]
\[\text{matrix}[3][0][0][180] = 1; \quad \text{// CNN} \]
\[\text{matrix}[3][0][0][37] = 1; \quad \text{// BitTorrent} \]
\[\text{matrix}[3][0][0][39] = 1; \quad \text{// AVI} \]
\[\text{matrix}[3][0][0][40] = 1; \quad \text{// Flash} \]
\[\text{matrix}[3][0][0][43] = 1; \quad \text{// QuickTime} \]
\[\text{matrix}[3][0][0][45] = 1; \quad \text{// WindowsMedia} \]
matrix[3][0][0][100] = 1;  // SIP
matrix[3][0][0][121] = 1;  // DropBox
matrix[3][0][0][124] = 1;  // Youtube
matrix[3][0][0][125] = 1;  // Skype
matrix[3][0][0][92] = 1;   // SSH
matrix[3][0][0][159] = 1;  // OpenVPN
matrix[3][0][0][163] = 1;  // TOR

matrix[3][1][0][2] = 1;   // POP3
matrix[3][1][0][3] = 1;   // SMTP
matrix[3][1][0][4] = 1;   // IMAP
matrix[3][1][0][65] = 1;  // IRC
matrix[3][1][0][119] = 1; // Facebook
matrix[3][1][0][120] = 1; // Twitter
matrix[3][1][0][122] = 1; // GMail
matrix[3][1][0][186] = 1; // FacebookChat
matrix[3][1][0][6] = 1;   // DNS
matrix[3][1][0][14] = 1;  // SNMP
matrix[3][1][0][17] = 1;  // Syslog
matrix[3][1][0][18] = 1;  // DHCP
matrix[3][1][0][81] = 1;  // ICNP
matrix[3][1][0][112] = 1; // LDAP
matrix[3][1][0][7] = 1;   // HTTP
matrix[3][1][0][70] = 1;  // Yahoo
matrix[3][1][0][123] = 1; // GoogleMaps
matrix[3][1][0][126] = 1; // Google
matrix[3][1][0][178] = 1; // Amazon
matrix[3][1][0][179] = 1; // eBay
matrix[3][1][0][180] = 1; // CNN
matrix[3][1][0][37] = 1;  // BitTorrent
matrix[3][1][0][39] = 1;  // AVI
matrix[3][1][0][40] = 1;  // Flash
matrix[3][1][0][43] = 1;  // QuickTime
matrix[3][1][0][45] = 1;  // WindowsMedia
matrix[3][1][0][100] = 1; // SIP
matrix[3][1][0][121] = 1; // DropBox
matrix[3][1][0][124] = 1; // Youtube
matrix[3][1][0][125] = 1; // Skype
matrix[3][1][0][92] = 1;  // SSH
matrix[3][1][0][159] = 1; // OpenVPN
matrix[3][1][0][163] = 1; // TOR

// Getter of the Output port for the first Switch. Return the Output port
A Code Developed

```java
// of the first Switch.
public int getOutputPortSwitch1(int IPs, int ProtocolID) {
    return matrix[0][IPs][0][ProtocolID];
}

// Getter of the Output port for the second Switch. Return the Output port
// of the third Switch.
public int getOutputPortSwitch2(int IPs, int IPd, int ProtocolID) {
    return matrix[1][0][IPd][ProtocolID];
}

// Getter of the Output port for the third Switch. Return the Output port
// of the second Switch.
public int getOutputPortSwitch3(int IPs, int ProtocolID, int retorno) {
    if (retorno == 0) {
        return matrix[2][IPs][0][ProtocolID];
    } else {
        return matrix[3][IPs][0][ProtocolID];
    }
}
```
package com.northbound.flowconstructor;

public class FlowModel {

  private String name;
  private String nwSrc;
  private String nwDst;
  // private String installInHw;
  // private String priority;
  // private String etherType;

  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

  public String getNwSrc() {
    return nwSrc;
  }
  public void setNwSrc(String nwSrc) {
    this.nwSrc = nwSrc;
  }

  public String getNwDst() {
    return nwDst;
  }
  public void setNwDst(String nwDst) {
    this.nwDst = nwDst;
  }
}

package com.northbound.flowreceiver;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import org.apache.commons.codec.binary.Base64;
import org.codehaus.jettison.json.JSONObject;

public class RestInterfaceReceiver {

    public JSONObject getNodes(String user, String password,
                                String baseURL) {
        StringBuffer result = new StringBuffer();
        try {
            if (!baseURL.contains("http")) {
                baseURL = "http://" + baseURL;
            }
            baseURL = baseURL + "/restconf/operational/opendaylight-inventory:"+ "nodes/node/openflow:1/flow-node-inventory:table/0/";

            // Create URL = base URL + container
            URL url = new URL(baseURL);

            // Create authentication string and encode it to Base64
            String authStr = user + ":" + password;
            String encodedAuthStr = Base64.encodeBase64String(authStr.getBytes());

            // Create Http connection
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();

            // Set connection properties
            connection.setRequestMethod("GET");
            connection.setRequestProperty(“Authorization”, "Basic " + encodedAuthStr);
            connection.setRequestProperty(“Accept”, "application/json");

            // Get the response from connection’s inputStream
            } finally {
                result.append(result.toString());
            }
        }
    }
}
InputStream content = (InputStream) connection.getInputStream();
BufferedReader in = new BufferedReader(new InputStreamReader(
    content));
String line = "";
while ((line = in.readLine()) != null) {
    result.append(line);
}

JSONObject nodes = new JSONObject(result.toString());
return nodes;
} catch (Exception e) {
    e.printStackTrace();
}

return null;
}
A.20 Rest Interface JSON Analyser Hydrogen

```java
package com.northbound.flowreceiver;

import org.codehaus.jettison.json.JSONArray;
import org.codehaus.jettison.json.JSONException;
import org.codehaus.jettison.json.JSONObject;
import com.northbound.flowconstructor.FlowModel;
import com.northbound.settings.Settings;

public class RestInterfaceJSONAnalyserHydrogen {

    public FlowModel[] flowsAlreadyInstalled() throws JSONException {
        RestInterfaceReceiver nodes = new RestInterfaceReceiver();
        JSONObject myFlows = nodes.getNodes(Settings.USERNAME, Settings.PASSWORD, Settings.URL);
        // System.out.println(myFlows);
        // Creation of a array of these Objects just to save every new flow and not install flows already installed.
        JSONArray array = myFlows.getJSONArray("flowConfig");
        FlowModel flowInArray[] = new FlowModel[55];
        for(int i = 0 ; i < array.length() ; i++){
            try{
                String nwSrc = array.getJSONObject(i).getString("nwSrc");
                flowInArray[i] = new FlowModel();
                flowInArray[i].setName(array.getJSONObject(i).getString("name"));
                flowInArray[i].setNwSrc(nwSrc);
                flowInArray[i].setNwDst(array.getJSONObject(i).getString("nwDst"));
                // System.out.println(flowInArray[i]);
            } catch(Exception e){
                continue;
            }
        }
        return compactingString(flowInArray);
    }

    public static FlowModel[] compactingString(FlowModel[] chainUnCompacted) {
        FlowModel[] compactedString = new FlowModel[chainUnCompacted.length];
        int j=0;
```
for (int i = 0; i < chainUnCompacted.length; i++)
    if (chainUnCompacted[i] != null) {
        compactedString[j] = chainUnCompacted[i];
        j++;
    }

return compactedString;
package com.northbound.flowreceiver;

import org.codehaus.jettison.json.JSONArray;
import org.codehaus.jettison.json.JSONException;
import org.codehaus.jettison.json.JSONObject;
import com.northbound.flowconstructor.FlowModel;
import com.northbound.settings.Settings;

public class RestInterfaceJSONAnalyserKaraf {

/**
 * @param args
 * @throws JSONException
 */
public FlowModel[] flowsAlreadyInstalled() throws JSONException {
    RestInterfaceReceiver nodes = new RestInterfaceReceiver();
    JSONObject myFlows = nodes.getNodes(Settings.USERNAME, Settings.PASSWORD,
                                  Settings.URL);

    FlowModel flowInArray[] = new FlowModel[55];
    JSONArray array = myFlows.getJSONArray("flow-node-inventory:table");
    JSONObject array2 = array.getJSONObject(0);
    JSONArray array_flow = array2.getJSONArray("flow");
    JSONObject flow = new JSONObject();

    for(int i = 0 ; i < array_flow.length() ; i++){
        try {
            flowInArray[i] = new FlowModel();
            flow = array_flow.getJSONObject(i);
            flowInArray[i].setName(flow.getString("id"));
            // System.out.println(flowInArray[i].getName());
            flowInArray[i].setNwSrc(flow.getJSONObject("match").
                                getString("ipv4-source").substring(0, 8));
            // System.out.println(flowInArray[i].getNwSrc());
            flowInArray[i].setNwDst(flow.getJSONObject("match").
                                getString("ipv4-destination").substring(0,8));
            // System.out.println(flowInArray[i].getNwDst());
        }
        catch(Exception e){
            continue;
        }
    }
}
A.22 Limit Lines Document Listener

```java
public class LimitLinesDocumentListener implements DocumentListener {
    private int maximumLines;
    private boolean isRemoveFromStart;

    public LimitLinesDocumentListener(int maximumLines) {
        this(maximumLines, true);
    }

    public LimitLinesDocumentListener(int maximumLines, boolean fromStart) {
        this.maximumLines = maximumLines;
        isRemoveFromStart = fromStart;
    }

    public void changedUpdate(DocumentEvent e) {
        if (isRemoveFromStart) {
            // Code to remove lines from the start
        } else {
            // Code to remove lines from the end
        }
    }

    public void flushUpdate(DocumentEvent e) {
        // Code to flush updates
    }

    public void insertUpdate(DocumentEvent e) {
        if (isRemoveFromStart) {
            // Code to remove lines when inserting at the beginning
        } else {
            // Code to remove lines when inserting at the end
        }
    }

    public void removeUpdate(DocumentEvent e) {
        if (isRemoveFromStart) {
            // Code to remove lines when removing from the start
        } else {
            // Code to remove lines when removing from the end
        }
    }
}
```

/*
* A class to control the maximum number of lines to be stored in a Document
* Excess lines can be removed from the start or end of the Document depending on your requirement.
* a) if you append text to the Document, then you would want to remove lines from the start.
* b) if you insert text at the beginning of the Document, then you would want to remove lines from the end.
*/
public LimitLinesDocumentListener(int maximumLines, boolean isRemoveFromStart) {
    setLimitLines(maximumLines);
    this.isRemoveFromStart = isRemoveFromStart;
}

/**
 * Return the maximum number of lines to be stored in the Document
 */
public int getLimitLines() {
    return maximumLines;
}

/**
 * Set the maximum number of lines to be stored in the Document
 */
public void setLimitLines(int maximumLines) {
    if (maximumLines < 1) {
        String message = "Maximum lines must be greater than 0";
        throw new IllegalArgumentException(message);
    }
    this.maximumLines = maximumLines;
}

// Handle insertion of new text into the Document
public void insertUpdate(final DocumentEvent e) {
    // Changes to the Document can not be done within the listener
    // so we need to add the processing to the end of the EDT

    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            removeLines(e);
        }
    });
}

public void removeUpdate(DocumentEvent e) {}
public void changedUpdate(DocumentEvent e) {}
private void removeLines(DocumentEvent e) {
    // The root Element of the Document will tell us the total number
    // of line in the Document.
    Document document = e.getDocument();
    Element root = document.getDefaultRootElement();
    while (root.getElementCount() > maximumLines) {
        if (isRemoveFromStart)
            removeFromStart(document, root);
        else
            removeFromEnd(document, root);
    }
}

private void removeFromStart(Document document, Element root) {
    Element line = root.getElement(0);
    int end = line.getEndOffset();
    try {
        document.remove(0, end);
    } catch (BadLocationException ble) {
        System.out.println(ble);
    }
}

private void removeFromEnd(Document document, Element root) {
    // We use start minus 1 to make sure we remove the newline
    // character of the previous line
    Element line = root.getElement(root.getElementCount() - 1);
int start = line.getStartOffset();
int end = line.getEndOffset();

try {
    document.remove(start - 1, end - start);
} catch(BadLocationException ble) {
    System.out.println(ble);
}
}
package com.northbound.gui;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;

/**
 * Create a simple console to display text messages.
 * Messages can be directed here from different sources. Each source can
 * have its messages displayed in a different color.
 * Messages can either be appended to the console or inserted as the first
 * line of the console
 * You can limit the number of lines to hold in the Document.
 */
public class MessageConsole
{
    private JTextComponent textComponent;
    private Document document;
    private boolean isAppend;
    private DocumentListener limitLinesListener;

    public MessageConsole(JTextComponent textComponent)
    {
        this(textComponent, true);
    }

    public MessageConsole(JTextComponent textComponent, boolean isAppend)
    {
        this.textComponent = textComponent;
        this.document = textComponent.getDocument();
        this.isAppend = isAppend;
        textComponent.setEditable( false );
    }
}
public void redirectOut()
{
    redirectOut(null, null);
}

public void redirectOut(Color textColor, PrintStream printStream)
{
    ConsoleOutputStream cos = new ConsoleOutputStream(textColor, printStream);
    System.setOut( new PrintStream(cos, true) );
}

public void redirectErr()
{
    redirectErr(null, null);
}

public void redirectErr(Color textColor, PrintStream printStream)
{
    ConsoleOutputStream cos = new ConsoleOutputStream(textColor, printStream);
    System.setErr( new PrintStream(cos, true) );
}

/*
 * To prevent memory from being used up you can control the number of
 * lines to display in the console
 * This number can be dynamically changed, but the console will only
 * be updated the next time the Document is updated.
 */
public void setMessageLines(int lines)
{
    if (limitLinesListener != null)
        document.removeDocumentListener( limitLinesListener );

    limitLinesListener = new LimitLinesDocumentListener(lines, isAppend);
    document.addDocumentListener( limitLinesListener );
}

/*
 * Class to intercept output from a PrintStream and add it to a Document.
 * The output can optionally be redirected to a different PrintStream.
 * The text displayed in the Document can be color coded to indicate
 * the output source.
 */
class ConsoleOutputStream extends ByteArrayOutputStream
{
    private final String EOL = System.getProperty("line.separator");
    private SimpleAttributeSet attributes;
    private PrintStream printStream;
    private StringBuffer buffer = new StringBuffer(80);
    private boolean isFirstLine;

    /*
    * Specify the option text color and PrintStream
    */
    public ConsoleOutputStream(Color textColor, PrintStream printStream)
    {
        if (textColor != null)
        {
            attributes = new SimpleAttributeSet();
            StyleConstants.setForeground(attributes, textColor);
        }

        this.printStream = printStream;

        if (isAppend)
            isFirstLine = true;
    }

    /*
    * Override this method to intercept the output text. Each line of text
    * output will actually involve invoking this method twice:
    * a) for the actual text message
    * b) for the newLine string
    * The message will be treated differently depending on whether the line
    * will be appended or inserted into the Document
    */
/**
 * public void flush()
 * {
 *     String message = toString();
 *     
 *     if (message.length() == 0) return;
 * 
 *     if (isAppend)
 *         handleAppend(message);
 *     else
 *         handleInsert(message);
 * 
 *     reset();
 * }
 */

private void handleAppend(String message)
{
    // This check is needed in case the text in the Document has been
    // cleared. The buffer may contain the EOL string from the previous
    // message.
    
    if (document.getLength() == 0)
        buffer.setLength(0);
    
    if (EOL.equals(message))
    {
        buffer.append(message);
    }
    else
    {
        buffer.append(message);
        clearBuffer();
    }
}

private void handleInsert(String message)
{

buffer.append(message);

if (EOL.equals(message))
{
    clearBuffer();
}

/*
 * The message and the newLine have been added to the buffer in the
 * appropriate order so we can now update the Document and send the
 * text to the optional PrintStream.
 */
private void clearBuffer()
{
    // In case both the standard out and standard err are being redirected
    // we need to insert a newline character for the first line only

    if (isFirstLine && document.getLength() != 0)
    {
        buffer.insert(0, "\n");
    }

    isFirstLine = false;
    String line = buffer.toString();

    try
    {
        if (isAppend)
        {
            int offset = document.getLength();
            document.insertString(offset, line, attributes);
            textComponent.setCaretPosition( document.getLength() );
        }
        else
        {
            document.insertString(0, line, attributes);
            textComponent.setCaretPosition( 0 );
        }
    }
    catch (BadLocationException ble) {}

    if (printStream != null)
    {
        printStream.print(line);
    }

    buffer.setLength(0);
A Code Developed
B.1 The GUI User Interface of the External Orchestrator
B.2 The Postman REST Client
B.2 The Postman REST Client
B.3 Flow in Hydrogen Release

```
<flow xmlns="urn:opendaylight:flow:inventory">
  <priority>10</priority>
  <flow-name>S1-S2</flow-name>
  <match>
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
    </ethernet-match>
    <ipv4-source>10.0.0.1/32</ipv4-source>
    <ipv4-destination>10.0.0.2/32</ipv4-destination>
  </match>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <output-action>
            <output-node-connector>2</output-node-connector>
          </output-action>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
</flow>
```

B.4 Flow in Helium/Lithium Release

```
<flow xmlns="urn:opendaylight:flow:inventory">
  <priority>10</priority>
  <flow-name>S1-S2</flow-name>
  <match>
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
    </ethernet-match>
    <ipv4-source>10.0.0.1/32</ipv4-source>
    <ipv4-destination>10.0.0.2/32</ipv4-destination>
  </match>
  <instructions>
    <instruction>
      <order>0</order>
    </instruction>
  </instructions>
</flow>
```
<apply-actions>
  <action>
    <order>0</order>
    <output-action>
      <output-node-connector>2</output-node-connector>
    </output-action>
  </action>
</apply-actions>
</instruction>
</instructions>
</flow>

B.5  Pom.xml

<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu_reina</groupId>
  <artifactId>L2Switch</artifactId>
  <version>0.1</version>
  <packaging>bundle</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.3.7</version>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Import-Package>
              *
            </Import-Package>
            <Export-Package>
              edu_reina.L2Switch
            </Export-Package>
            <Bundle-Activator>
              edu_reina.L2Switch.Activator
            </Bundle-Activator>
          </instructions>
          <manifestLocation>${project.basedir}/META-INF</manifestLocation>
        </configuration>
      </plugin>
    </plugins>
<dependencies>
  <dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>sal</artifactId>
    <version>0.7.0</version>
  </dependency>
</dependencies>

<repositories>
  <!-- OpenDaylight releases -->
  <repository>
    <id>opendaylight-mirror</id>
    <name>opendaylight-mirror</name>
    <url>http://nexus.opendaylight.org/content/groups/public/</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
  </repository>
  <!-- OpenDaylight snapshots -->
  <repository>
    <id>opendaylight-snapshot</id>
    <name>opendaylight-snapshot</name>
    <url>http://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/</url>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>
Bibliography


[13] Diego Kreutz, Member, IEEE, Fernando M. V. Ramos, Member, IEEE, Paulo Verissimo, Fellow, IEEE, Christian Esteve Rothenberg, Member, IEEE, Siamak Azodolmolky, Senior Member, IEEE, and Steve Uhlig, Member, IEEE.


Acknowledgement

The completion of this dissertation could not have been possible without the documents and information written by so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. However, I would like to express my deep appreciation and indebtedness particularly to the following:

To my family,
the people that has helped me with it, not only with their assistance but also with their support: Andrepi, Walter, Franco, Nour, Francesco, Chiara;

to my country, Catalonia
'Si jo l’estiro fort per aquí
i tu l’estires fort per allà,
segur que tomba, tomba, tomba,
i ens podrem alliberar.’ Lluis Llach.

and especially to Claudia.

I thank you.

Eduard Reina Fuente