A Carrier Grade OpenFlow Controller based on the JAIN SLEE Component Model

Jhon Jaime Martinez Alegria

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

Jhon Jaime Martinez Alegria

A Carrier Grade OpenFlow Controller based on the JAIN SLEE Component Model

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Flavio Oliveira Da Silva

Uberlândia
2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

M385c
2017

Martinez Alegria, Jhon Jaime, 1991-
A carrier grade OpenFlow controller based on the JAIN SLEE component model / Jhon Jaime Martinez Alegria. - 2017.
77 f. : il.

Orientador: Flávio de Oliveira Silva.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Rede definida por software (Tecnologia de rede de computadores) - Teses. 3. Controladores programáveis - Teses. 4. - Teses. I. Silva, Flávio de Oliveira, 1970-. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Este trabajo esta dedicado a mi familia, que me brindo el apoyo y soporte para avanzar un paso mas en la excelencia.

Agradecimentos

Agradeço a todos que me deram uma ajuda direta ou indireta para conseguir alcançar meus objetivos, Caio Thomas por toda a ajuda que eu recebi na faculdade desde que cheguei para o Brasil, Caio Ferreira porque ele liderou a construção de um excelente projeto e também um amigo que estava sempre lá para me brindar ajuda quando foi necessário, meu professor Flavio quem foi meu orientador nesta etapa academica da minha vida é a Capes, fomentadora crucial no desenvolvimento do meu mestrado.

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
(Eleanor Roosevelt)

Resumo

As Redes Definidas por Software (SDN) e Virtualização de Funções de Rede (NFV) estão atraindo a atenção dos operadores de telecomunicações. O ambiente de telecomunicações requer um conjunto de requisitos de *Carrier Grade*, como alta disponibilidade, alto rendimento e baixa latência. O SDN enfrenta desafios para atender o *Carrier Grade*. Baseado no modelo de componente JAIN SLEE e no *OpenFlow* driver Libfluid, este trabalho propõe o controlador SDN CREDENCE. Nossa avaliação mostra que o controlador CREDENCE oferece um throughput maior e uma menor latência quando e comparado ao ONOS e OpenDayLight..

Palavras-chave: SDN.OpenFlow.LibFluid.Carrier Grade.Controller.

Jhon Jaime Martinez Alegria

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA – UFU
FACULDADE DE COMPUTAÇÃO – FACOM
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO – PPGCO

The undersigned hereby certify they have read and recommend to the PPGCO for acceptance the dissertation entitled **"A Carrier Grade OpenFlow Controller based on the JAIN SLEE Component Model"** submitted by **"Jhon Jaime Martinez Alegria"** as part of the requirements for obtaining the **Master's degree in Computer Science**.

Uberlândia, 01 de Março de 2017

Supervisor: ______________________________
Prof. Dr. Flávio de Oliveira Silva.
Universidade Federal de Uberlândia

Examining Committee Members:

____
Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

____
Prof. Dr. Daniel Nunes Corujo
Instituto de Telecomunicações, Campus Universitário de Santiago - Aveiro, Portugal

Abstract

Software Defined Network (SDN) and Network Function Virtualization (NFV) are attracting the attention of telecom operators. The telecom environment requires a set of *Carrier Grade* requirements such as high availability, high throughput and low latency. SDN faces challenges to meet the *Carrier Grade*. Based on the JAIN SLEE component model and the Libfluid *OpenFlow* driver, this work proposes the CREDENCE SDN controller. Our evaluation shows that CREDENCE controller offers a higher throughput and a lower latency when is compared to ONOS and OpenDayLight.

Keywords: SDN.OpenFlow.LibFluid.Carrier Grade.Controller.

List of Figures

List of Tables

Acronyms list

API	Application Programming Interface
CREDENCE	Carrier Grade Software Defined Networking Control Environment
FTP	File Transfer Protocol
JAIN	Java APIs for Integrated Networks
LDAP	Lightweight Directory Access Protocol
NFV	Network function virtualization
NBI	North Bound Interface
ONF	Open Networking Foundation
RA	Resource Adapter
REST	Representational State Transfer
SDN	Software Define Network
SLEE	Service Level Execution Enviroment
SBB	Service Building Blocks
SNMP	Simple Network Management Protocol
SYSLOG	System Log
TACACS	Terminal Access Controller Access Control System
XML	Extensible Markup Language

Contents

CHAPTER 1

Introduction

The Software Define Networking (SDN) is the topic of network technology with most attention the last year because it has faster innovation, enabling programmability and allow easier management, all this one is possible by its paradigm to separate the data plane to the control plane and be controlled from one central entity (DIXIT et al., 2013) (SHALIMOV et al., 2014). The decisions of the forwarding are done first in their controllers and then are sent to the data plane which only execute those instructions .

The bigger growth of the requirements to the networks and the little advance in the level of network hardware, has left a bottleneck for the progress in the integration of services (MCKEOWN et al., 2008). Environments such as telecommunications daily require innovations to align with the speed of growth which motivates the fast adoption of SDN. *Software Define Networking*, a network technology has the potential of innovation and capacity to ensure the network operators have more control of their infrastructure and activate *OpenFlow*. A protocol embodying some concepts, SDN allows network signaling in the control layer, which permits controllers to manage network devices compatible with *OpenFlow* in a programmable way. It is now possible to find new ways to increase the performance and use of existing elements, so as to obtain a continuous improvement, focusing on the controllers and even more as data controllers, complying the requirements of each customer at a time when the communication demands high quality in the provision of services in addition to a higher capacity, security and low latency.

SDN abstractions can be realized by the *OpenFlow* switch. In a *OpenFlow* capable switch, the forwarding plane behavior can be driven by the SDN controller using the *OpenFlow* protocol (MCKEOWN et al., 2008). The SDN controller uses the protocol to modify the *Flow Table* entries in the switch. After a match on a key in the table, each packet is forwarded by a corresponding action.

Telecom operators equipments and systems must comply to several requirements such as high availability, high throughput, low latency, security and management. This set of requirements are often described as *Carrier Grade* (Intel Embedded Community, 2014). In this way, to be used on the telecom environment *OpenFlow* controllers must be *Carrier*

Grade.

There are several different controllers and most of them are open and suitable to the research environment (SALMAN et al., 2016). Controllers like ONOS (ONOSPROJECT, 2014), OpenDayLight (MEDVED et al., 2014) and Ryu (RYU, 2011) are the most used controllers in the current SDN market (SDNCENTRAL, 2016). SDN and also SDN controllers face several challenges to provide resiliency (MARTINO; MENDIRATTA; THOTTAN, 2016). Controllers suffers with issues such as data consistency (HAN; LEE, 2016) and fail over time versus the number of controlled devices in clustered deployments (SUH et al., 2016).

In this work we build a *Carrier Grade OpenFlow* controller which takes advantage of a software platform that is already deployed in the telecom industry and offers high throughput, low latency, scalability (FEMMINELLA et al., 2009) and availability (Femminella et al., 2011). This platform is called *JAIN SLEE*. Instead of creating a controller from scratch, our controller uses the the event driven component model provided by *JAIN SLEE* and the Libfluid *OpenFlow* driver (VIDAL; ROTHENBERG; VERDI, 2014).

The result is a controller that presented better throughput and latency results in the benchmarking test among the main controllers of the market also with good network size performance and that is able to be implemented in an SLEE environment. The library that was implemented is *LibFluid* (Vidal; ROTHENBERG; VERDI, 2014), a library chosen by the *ONF* to be the default library in its main features. Joining the controller and the library already developed together with some extra modules helps to maximize the use altogether, and allows an improvement in the current state of the controller, in order to meet the requirements that a telecommunications environment may demands.

1.1 Motivation

In a previous work, inside the project *Carrier Grade Software Defined Networking Control Environment (CREDENCE)* our research group started an investigation aimed to the development of a control layer for SDN that should meet carrier grade requirements such as performance, availability and scalability. This research was based on the use of a carrier grade service level execution environment based on the JAIN SLEE specification , capable of providing a cross layer platform that can satisfy the requirements from telecom operators (FERREIRA et al., 2014). A central component of this control layer was a *OpenFlow* controller. The proof of concept controller was created by wrapping Floodlight (DIXIT et al., 2013) *OpenFlow* controller inside the JAIN SLEE component model.

This proof of concept controller only supports *OpenFlow* 1.0 protocol and needs to be able to support the *OpenFlow* 1.3 protocol, also include new capabilities to the controller capable of servicing new services that can be implemented inside the JAIN SLEE component model and connected to the controller. All of the above with a Carrier Grade approach that allows the implementation in a real telecom environment.

1.2 Thesis Organization

The first part of this chapter is the introduction: basic principles, history, and motivation of this thesis. **Chapter 2** provides an overview of the state of art where we compiled the basic information which was taken as the basis for the solution of this project. **Chapter 3** discusses the credence architecture and all the elements that were implemented throughout the work. **Chapter 4** includes some benchmarks that were made to validate the installation and perform the comparison with some active controllers. **Chapter 5** Discusses the project's findings as well as other possible advances in the investigation related to SDN Controller.

Chapter 2

Background

This chapter will present an approach to the main definitions of the terms used throughout the project, as well as the definitions of the protocols, techniques and libraries used for the development of the controller. To conclude the chapter, is included a comparison of controllers that allowed us to focus the creation of the controller to the current needs.

2.1 Carrier Grade Requirements

In order to have a first approximation to the Carrier Grade Requirements we want to make the term Carrier Grade clear. A term that is used widely in the public network telecommunications products, which specifies a higher percentage at 99.999% of availability, the same is usually associated with carrier-class servers or carrier-class switches (Intel Embedded Community, 2014).
Currently, software systems that control networks guarantee a "six nine" reliability, making a few calculations that approaches to 32 seconds per year of potential downtime, and this is not only a level configurations *Carrier Grade* (RCR Wireless, 2015), there are 4 categories that should be detailed before continuing with the description as shown in the Figure 1.

Starting with the availabilities of which concerned the issue arises, a telecommunications network needs to have redundancy in terms of 500km, to be sufficiently capable of detecting errors or faults, and recovering quickly in a time of no more than 500ms (RCR Wireless, 2015).
In a second area of interest network security is discussed a little beyond the security systems that companies can handle to manage data. In a telecommunication network there should not be a traffic that is observable and the same must be encrypted. In addition the customer information should not be stored during the applications time. The system must be capable of providing security protocols (authentication, authorization and accounting), preventing unauthorized access.

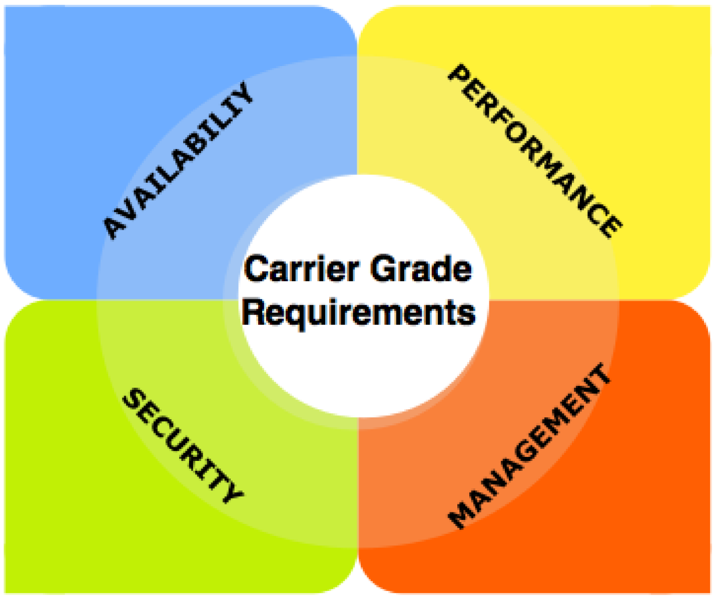

Figure 1 – Carrier Grade Requirements

For a third area of interest, there is a part about the performance; as the network must deliver high performance, but in turn handle low latency to ensure the reliability of critical real time applications, giving the example of a migration of virtual machines for load balancing, you should be in no more than 150ms cutting.Finally in a fourth category, the system must be compatible updates "on fire", supporting patch deployment without interruption in the flow system.
Based on the above categories it is easily denoted that in the telecom context the implementation of a system that complies with the requirements is necessary to be able to offer a greater capacity and availability of service, as well as allowing companies a quality approach.

2.2 Jain Slee

Service Level Execution Enviroment (SLEE) is a term well known in the telecommunications industry. Slee is a service execution environment with low latency, high performance and event driven asynchronously, *JAINSLEE* is also the standard Java-Slee environment designed for deployments where are possible to achieve scalability and availability through cluster architectures, standards *JAINSLEE* allows implementations that meet the requirements of large communications applications.
The objective of Java APIs for Integrated Networks (JAIN) is to provide service portability, converge and secure access to such integrated networks. As a point of integration of resources by multiple networks and protocols, applications can use external resources

along with JAINSLEE environment, allowing developers to create robust components and ACID type solutions (PELTZ, 2003).
Its philosophy is defined as a collection of reusable object-oriented components and containers that will host these components in real time, and support the development of application servers highly available and scalable.
SLEE includes the control interfaces for the administration of the application environment and the application components executed in the system. Within the system administrator, JAINSLEE controls the service life cycle through the standard management interfaces given by the compliant inside this, and it includes the characteristic information of the service like the name, the version, the vendor and other Information that could be part of the service. Inside the service may be included Java classes, profiles and Service Building Blocks (SBB).
The JAINSLEE profile contains the data provided based on a set of attributes or properties that can be modified using the management interfaces that the profiles include, in order to define a profile schema and its logic. Therefore the SBB are able to access this information as part of their logical application.
The SBB or Service Building Blocks are blocks of instructions in a software component that sends and receives events besides performing the computational logic based on the receipt of these events and their current state. Each SBB is able to communicate with other service blocks and may be formed in turn by other SBBs, this subject will be further discussed in the section 2.2.3.

2.2.1 Mobicents

The Mobicents Communication Platform is an architecture to create, deploy and manage services and applications integrating voice, video and data across a range of IP and legacy communications networks (MOBICENTS.ORG, 2016) and incluide a SLEE that makes use of the JBoss. Jboss offers capabilities for service and SLEE management, service deployment and thread pooling in this SOA architecture with GNU license.
The access to protocols and network elements is provided by an interface layer called Resource Adaptor, and they provide a set of interfaces to be used in the SBB.

2.2.2 Resource Adaptor (RA)

JSLEE is an application server, which is based on components, and the events in the JSLEE application model are Plain Old Java Objects (POJOs) and they need to be created somewhere, which is why the RA exists.The Resource adaptor in JSLEE bridges the application model with the data plane model that receives specific events and creates the java representations to fire them into the JSLEE application server.

According to the *JAINSLEE* specification, a *resource* represents a system that is external to a SLEE, and the SLEE architecture defines how the application running within the SLEE interact with resource through resource adaptors.

The Resource Adaptor is a SLEE component which is used for communicating between the SLEE and an external resource. In a project for example, the Resource Adaptor binds to a network socket using the controller and translate incoming network messages into events which are then delivered to the SLEE. These events would then drive logic in a service, which could then send messages back to the network via Resource Adaptor. The Resource Adaptor has the following principal concepts (FERRY, 2008):

- ❏ The Resource Adaptor Entity who is an instantiation of a Resource Adaptor implementation.
- ❏ The Resource Adaptor Type declares the common characteristics for a set of resource adaptors.
- ❏ The Resource Adaptor is an implementation of a one or more resource adaptor types.
- ❏ The Resource Adaptor Object who is a Java Object created from a Resource Adaptor implementation for the purpose of executing Resource Adaptor Logic
- ❏ SLEE endpoint, is the interface between the resource adaptor and the SLEE

Understanding these terms, is a little clearer operation of Resource Adapter (RA) within the project because inside this, multiple instances of the Resource Adaptor Entity are created and within the entire logic is processed by the Resource Adaptor Object. Without forgetting that each object created within the Resource Adaptor requires a context,which is assigned in the creation of the object allowing access to reference data in the current context.
The Resource Adapter is a key part of the project, working as a bridge between the controller and the events that will be created as well as the callbacks presented during the project. This requires activating a listener and a trigger event, on one hand the listener, the networks receive messages and in the other side, the pitcher trigger events by SBB.

2.2.3 Service Building Blocks (SBB)

The elements used by *JAINSLEE* for reusing are called SBB *(Service Building Blocks).* SBB is an application component that sends and receives events and performs computational logic based on the status of the event and your call. Each SBB identifies the event types accepted by the component and it has event handler methods that contain application code that processes events of these event types.
The state of the entities of SBBs can be replicated in a clustered deployment. Using

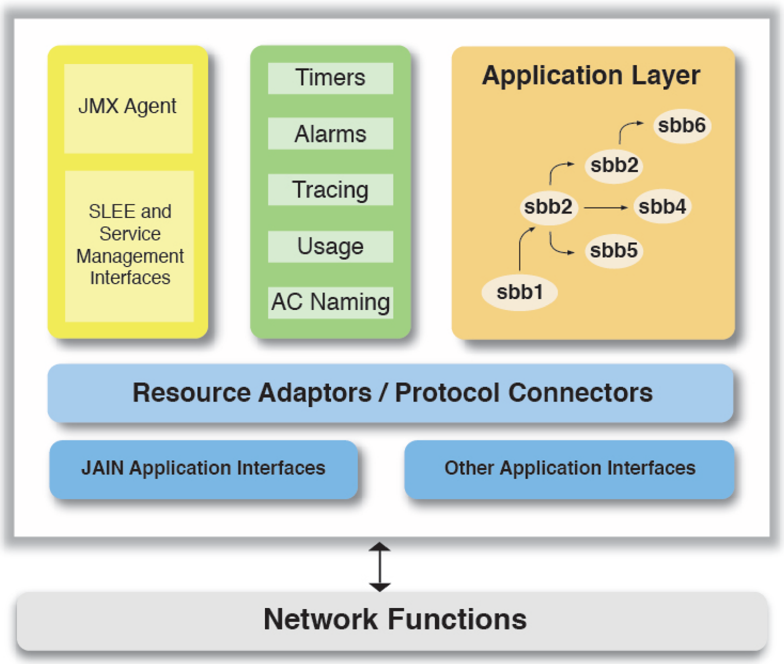
JMX Agent
SLEE and Service Management Interfaces
Timers
Alarms
Tracing
Usage
AC Naming
Application Layer
sbb6
sbb2
sbb2
sbb4
sbb5
sbb1
Resource Adaptors / Protocol Connectors
JAIN Application Interfaces
Other Application Interfaces
Network Functions

Figure 2 – JAIN SLEE Architecture Main Components

also the characteristics that it has JAIN SLEE where it can support high availability and fault tolerance it is possible to use clustering, which means that all services that were previously implemented in the upper layers can be deployed as SBBs.
An SBB can be formed by other SBB, enabling the interpretation of problems in more complex applications that are based on the logic of the receipt of events, In turn a part of its components is the RA, which was an external entity to interact with other external systems and protocols stacks SLEE, directories and databases, as shown in Figure 2.

2.3 Software Defined Networking

Software Define Networking is a set of techniques that facilitate the implementation of network services in a more dynamic and scalable way. Everything is due to the separation of the orchestration layers where it consists of decoupling the control and data plane of a network.
SDN enables management of the Control plane through software. Carrying out this separation makes possible to use software components to perform the network control plane (KREUTZ et al., 2015). These components are called SDN controllers or controllers.The data plane continues being managed, by connectivity equipment like the Figure 3.

Network Functions Virtualization (NFV) aims to consolidate many network equipment types onto commodity high volume servers, switches and storage using virtualization techniques enabling the creation of virtual appliances (CHIOSI DON CLARKE, 2012).

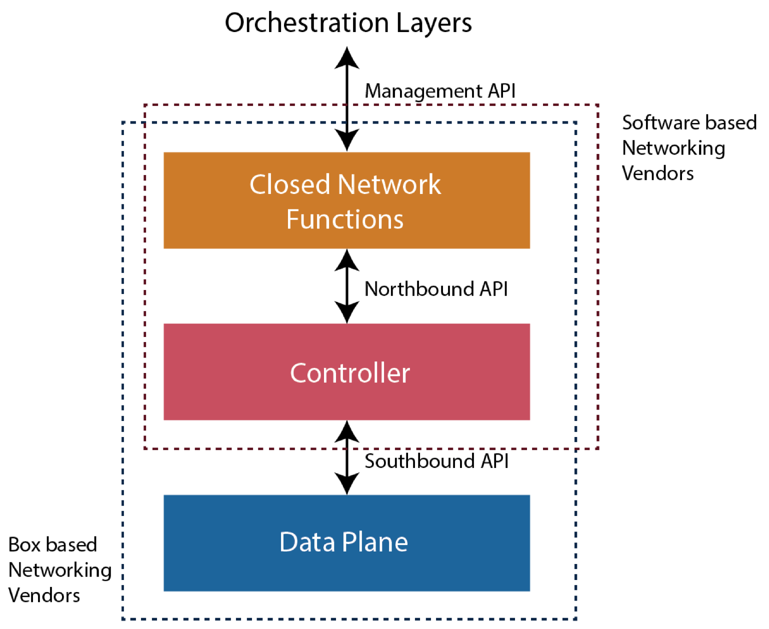

Figure 3 – SDN and Traditional Network Platform Architecture (MONCLUS, 2013)

Adopted in a complementary way, NFV and SDN may enable significant reductions in operating expenses (OPEX) and capital expenses (CAPEX) and facilitate the deployment of new services by increasing the agility of deployment providing a faster time-to-value (MIJUMBI et al., 2016). These technologies are attracting the attention of network operators (WARWICK, 2015).

In the present the selection of controllers has increased since daily new controllers are signed to cover certain needs in the evolutionary stage of network communication that are being presented, most don't maintain a uniform style programming language, as the first controllers were designed in C++ and some others use Python or ava (NICIRA, 2011), (OPENFLOW.STANFORD.EDU, 2015), (KOPONEN et al., 2010), (GOOGLE, 2015), but all with special needs.

With the advancement of each controller it is required to improve many aspects, such as making a stronger high availability, scalability and having a better performance, others focus turn to cover all general needs and it's there where the focus of the project is underway. Projects like ONOS (BERDE et al., 2014) and OpenDayLight project (MEDVED et al., 2014) are the latest and the research ground of this project.
The *(Open Network Operating System)* ONOS project (ONOSPROJECT, 2014) is still in development but it offers alternative communication. Providers service was introduced in 2013 and in 2015 their second prototype was released, focused on improving performance, reducing latency and adding a notification framework, in addition to modify some data model by adding a layer cache for faster access to storage thereof (ON.LAB, 2013).
In this scenario, ONOS emerges as a experimental controller platform that provides a distributed, but logically centralized, global network view, scale-out and fault tolerance

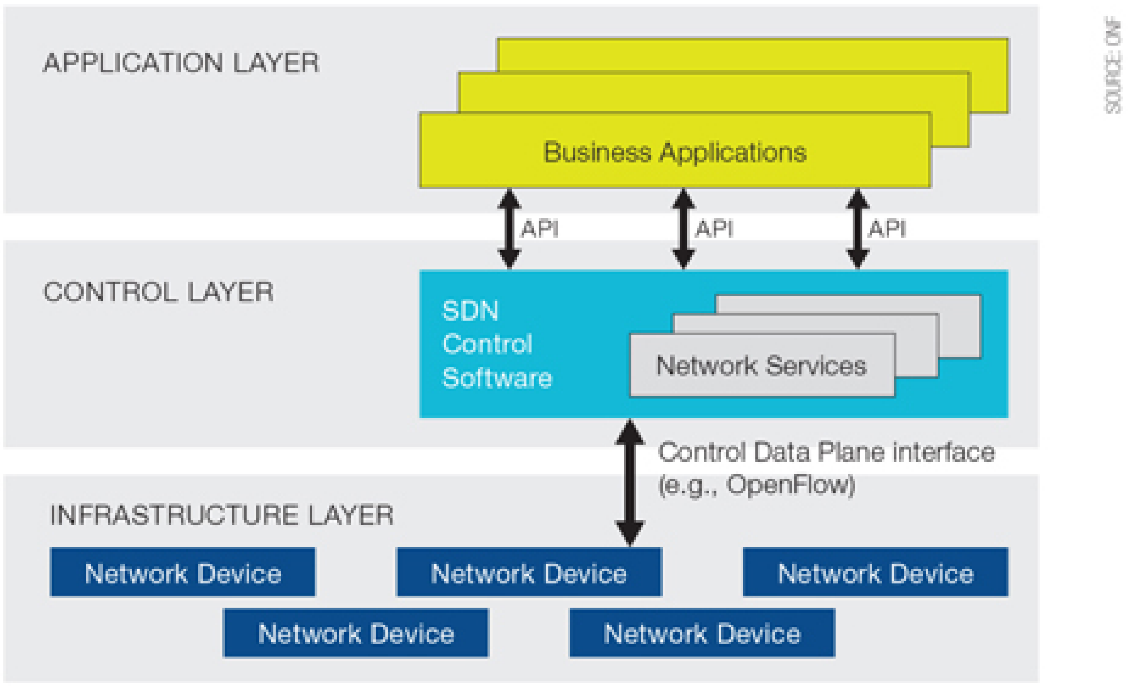

Figure 4 – SDN Architecture (SDXCENTRAL, 2015)

by using a consistent store for replicating application state, all with *Carrier Grade* focus. The *OpenDayLight* project is also an open operating system of Linux Foundation, which is focused on open programming for creating SDN and Network function virtualization (NFV) (PITT SEDEF OZCANA, 2014) that are scalable to any size. The project is managed through service layers of abstraction that allows communication between different protocols and offers extensive control through an API that can be used by higher application layers.

2.4 OpenFlow

OpenFlow is a switching technology that was created in the root of the research project at Stanford University (MCKEOWN et al., 2008). It is defined as an emerging and open communications protocol that allows an application server to determine the packet forwarding path that should be followed in a network of switches.
The protocol allows network management, establishing decisions involving the movement of centralized packages, thus being configurable regardless of the switches. In conventional switches, the data layer and the control layer were run on the same device, but thanks to the implementation of *OpenFlow* and SDN they are separated and so routing high level decisions are taken by a controller and communicated to the switch through *OpenFlow* like the Figure 4.

The methodology allows for more effective use of resources against a conventional net-

work, with the possibility of permitting researchers to test switches with heterogeneous line speed and high port density, as could happen in an environment of telecommunications, existing as a possible node, each subscriber of the services they provide (DAS; PARULKAR; McKeown, 2012).
It is in the telecommunications environment where this project is focused and with the help of innovations on the handling of packets on a network, new requirements and very great opportunities to take advantage of this research are created. Current research focuses on all possible layers of SDN architecture turn few research groups, centered on validating the main requirements that demands a telecommunications environment as previously commented on *Carrier Grade*. Researches like ONOS(BERDE et al., 2014) who are working on these type of specifications, have many solutions to many problems, in turn, they also have many weaknesses which have not yet been resolved.
Still, the controllers needs to be updated with new tools and protocol elements which have arisen since it was created, and the implementation of libraries for parsing data and modules to strengthen the provision of services, to improve the current situation and offer a generic, stable and optimized version for the new requirements of telecommunications network issues.

2.5 LibFluid

Libfluid, a low-overhead library, portable and configurable SDN control connectivity layer that serves different use cases with a very basic set of tools (using the *OpenFlow* protocol as a mean). It uses generic properties and functions capable of being understood by the switches and controllers, being a unified connectivity layer, helping to improve the abstraction of the system, besides it also uses a low level abstractions allowing to be lighter on the load the CPU, and in turn, it is compiled C++ language, which in addition, reduces the final weight of the code being executed and permits low-level abstraction. It controls a protocol agnosticism avoiding version compatibility issues between versions of the protocol and only focusing on the basics, allowing greater flexibility.
In the libfluid structure, two important parts are centered, on the one hand libfluid_msg and on the other libflud_base, each one is responsible for some functionalities. Starting with libflud_base, where the *OFServer* operates and works as the bridge between the communication and the logic of the controller being used. Libfluid_msg in addition to include the msg callback, basically performs the parsing of the *OpenFlow* messages in order to create objects for each *OFType* received, thus, it is easier to build the logic in the base of the library and that can consume all the Services from a client server architecture. The operation of libfluid is based on a client-server architecture, where the client is switch and the server is the controller as shown in the Figure 5. In this way, elements and connections are connected by libfluid_base and sent right through the channels of commu-

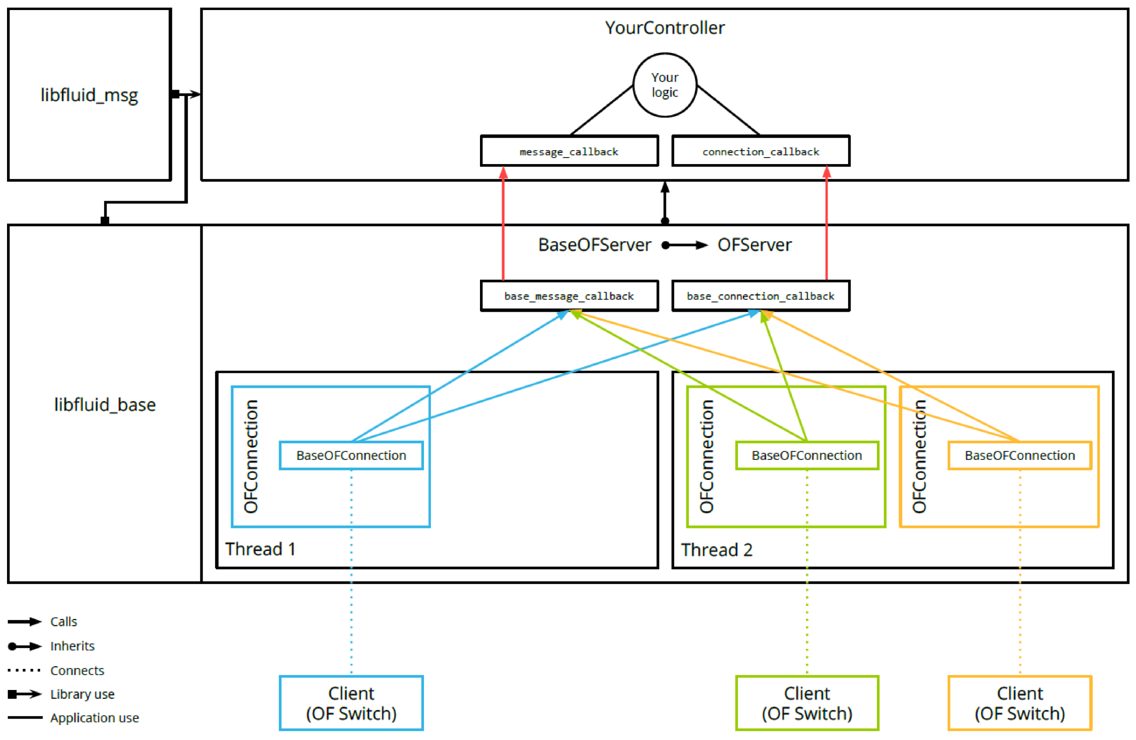

Figure 5 – libfluid architecture(Vidal; ROTHENBERG; VERDI, 2014)

nication and serviced by libfluid_msg, which handles data parsing creating works with system events.

2.6 OpenFlow Controllers

Today there are many options for SDN controllers, some newer than others and in particular each focused on a specific solution, written in a different programming language. For the project the focus will be on the two main Controllers: *ONOS* and *OpenDayLight*, because they are those with a focus of application in production environments and also postulated as *Carrier Grade* Controllers. Which resembles the main intention of the project and can be part of the comparison in the testing area.

2.6.1 ONOS

The Open Network Operating System (ONOS) is the first open source SDN network operating system targeted specifically at the service provider and mission critical networks. ONOS is purpose built to provide the high availability (HA), scale-out, and performance these networks demand. In addition, ONOS has created useful Northbound abstractions and that enable easier application development and Southbound abstractions and interfaces to allow for control of *OpenFlow* ready and legacy devices (BERDE et al., 2014).

In this scenario, ONOS *Open Network Operating System*, emerges as the controller of choice for such networks and attends the criteria that qualify as *Carrier Grade* (SILVA et al., 2014). These criteria are high availability, reliability and fault tolerance. In other words, the ONOS was developed with necessary features so that it can be used by telecommunications companies.
ONOS is a project developed by the team of ON.Lab, a non-profit organization specializing in SDN and *OpenFlow* technologies. The organization is supported by big names in the telecommunications market and computing, such as At&T, Cisco, Ciena, Fujitsu, Huawei, Intel, NEC and the Open Networking Foundation (ONF).
As Figure 6 illustrates, the ONOS is divided into five layers:

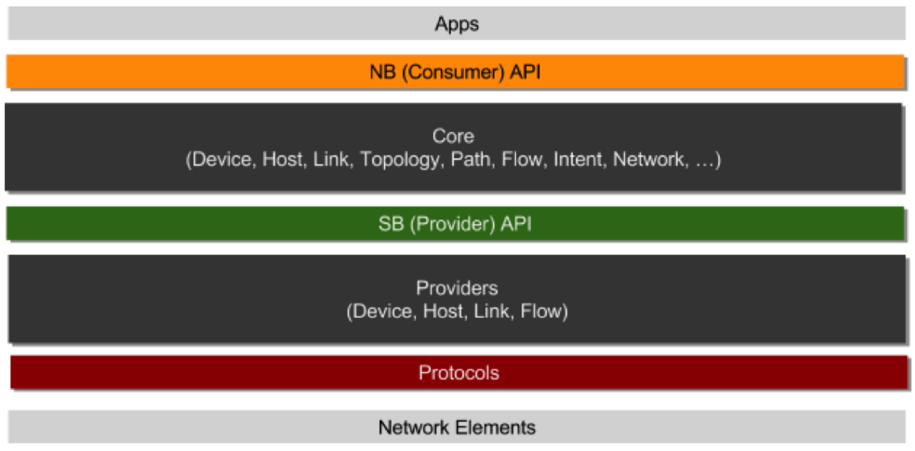
Apps
NB (Consumer) API
Core
(Device, Host, Link, Topology, Path, Flow, Intent, Network, ...)
SB (Provider) API
Providers
(Device, Host, Link, Flow)
Protocols
Network Elements

Figure 6 – Onos Architecture (ONOSPROJECT, 2014)

- ❑ Distributed Core: it is the main layer, performing the management of resources and the status of the network as a whole and for managing the distributed execution of the system, bringing the functions of *Carrier Grade* to carrier grade control plane. The ability of Onos run as a *cluster* is the way that makes it quick to meet the control plane and telecom operators it needs.

- ❑ Northbound abstraction/API: This allows the development applications that use Onos as a platform, such as the graph of the network topology. The Internet allows Frameworks applications request services without knowing how they were made.

- ❑ Southbound abstraction/API: The layer is composed of modules that perform control and direct access to each device. The providers of the API abstract each device, allowing all of them to be used in the same way in the upper layers.

- ❑ Modularity Software: Makes development easier, debugging, maintenance and updating Onos and as well as the programs developed by the community and suppliers.

2.6.2 OpenDaylight

OpenDayLight is an Open Source SDN project managed by the Linux Foundation and founded in the company of other companies like Arista Networks, Big Switch Networks, Brocade, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Nuage Networks, PLUMgrid , Red Hat and VMware in 2013. Its main focus is to accelerate the process of adaptation of SDN, besides being created in conjunction with research groups in general.
The source code is written in java and to date four releases have been launched. Hydrogen in February 2014 and ending with Beryllium with a renewed structure compared to previous versions 7 in October 2016. The edition of the code is given by 3 levels of membership being Platinum, gold and silver. In platinum there are companies like Cisco, Intel, and Dell. Companies like NEC in gold and in silver, companies like AT&T, Fujitsu, IBM, Lenovo and others.

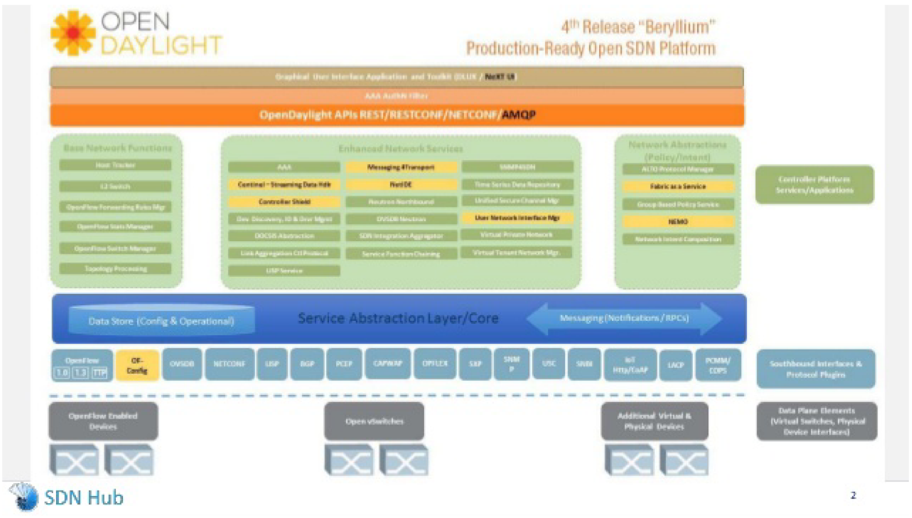
OPEN DAYLIGHT
4th Release "Beryllium"
Production-Ready Open SDN Platform
Graphical User Interface Application and Toolkit (DLUX / NeXT UI)
AAA AuthN Filter
OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP
Base Network Functions
Host Tracker
L2 Switch
OpenFlow Forwarding Rules Mgr
OpenFlow Stats Manager
OpenFlow Switch Manager
Topology Processing
Enhanced Network Services
AAA
Centinel – Streaming Data Hdlr
Controller Shield
Dev Discovery, ID & Drvr Mgmt
DOCSIS Abstraction
Link Aggregation Ctl Protocol
LISP Service
Messaging4Transport
NetIDE
Neutron Northbound
OVSDB Neutron
SDN Integration Aggregator
Service Function Chaining
SNMP4SDN
Time Series Data Repository
Unified Secure Channel Mgr
User Network Interface Mgr
Virtual Private Network
Virtual Tenant Network Mgr.
Network Abstractions (Policy/Intent)
ALTO Protocol Manager
Fabric as a Service
Group Based Policy Service
NEMO
Network Intent Composition
Controller Platform Services/Applications
Data Store (Config & Operational)
Service Abstraction Layer/Core
Messaging (Notifications / RPCs)
OpenFlow 1.0 1.3 TTP
OF-Config
OVSDB
NETCONF
LISP
BGP
PCEP
CAPWAP
OPFLEX
SXP
SNMP
USC
SNBI
IoT Http/CoAP
LACP
PCMM/COPS
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)
SDN Hub
2

Figure 7 – Beryllium (OPENDAYLIGHT, 2016)

OpenDayLight includes support for the *OpenFlow* protocol, but also supports NFV (YUE, 2013) standards. The controller can support a modular controller framework in the low layers. In the northbound layer it exposes an open API that can be used by applications.
With the API, users can access to information gathering, run algorithms and create rules for sending messages on the network. OpenDayLight contains its own java virtual machine which makes it a valid option that can be deployed on operating system platforms with java support.

2.6.3 PoC SLEE Controller

The controller was developed under the JAINSLEE specification and by referencing the operation of Mobicents. Mobicents is also an open source implementation created

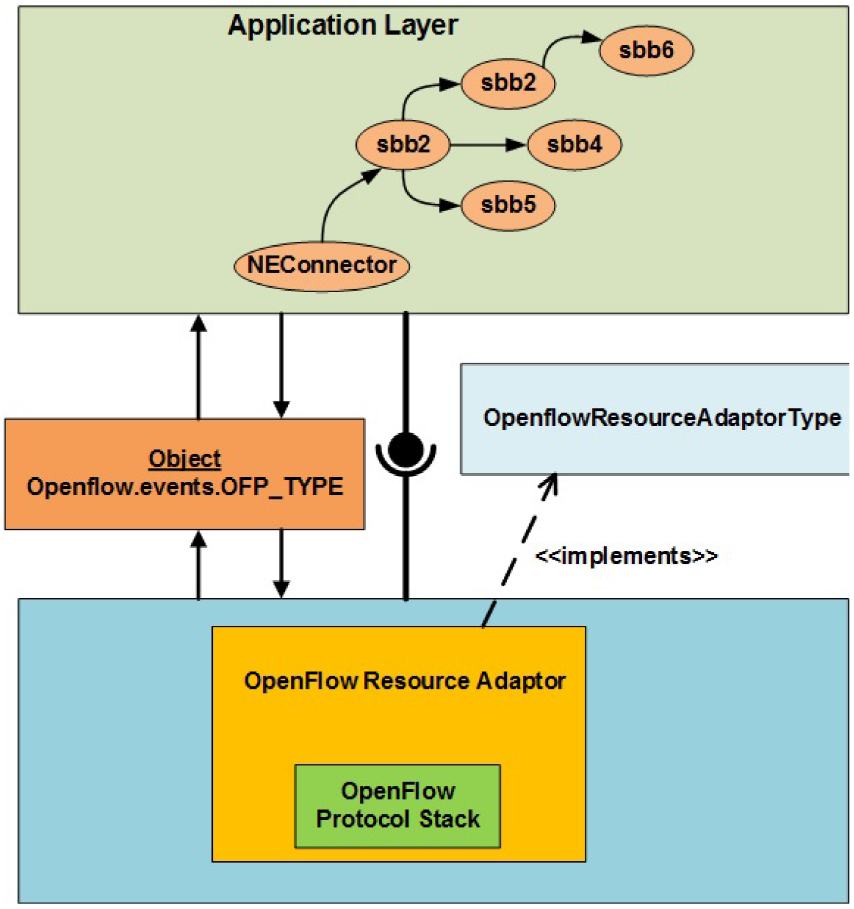

Figure 8 – Open Flow Resource Adapter(FERREIRA et al., 2014)

under the specification of JAINSLEE, and it was used for the analysis of the construction of a RA.

The RA was created for the *OpenFlow* 1.0 protocol and it does not follow the implementation requirements of the JAINSLEE component model to support fault tolerance. In order to prove the concept, the parsing of *OpenFlow* messages was done by using the FloodLight controller (DIXIT et al., 2013) that was wrapped into the RA.

Its operation was based on the reception of messages by the RA to the SBB where RA consumes events created by the services within JAINSLEE and transmits to switches equipped with *OpenFlow* as shown in the Figure 8. So was it possible to create a controller that was based on JAINSLEE component model in a straightforward way. However, performance and also fault tolerance was not the focus of this *OpenFlow* controller.

Thus, implementing the library and accompanying modules, it is possible to improve the current state of controller performance, as other controllers using the example of OpenDayLight (MEDVED et al., 2014) and floodlight (DIXIT et al., 2013).

Table 1 – Controller Comparative

	ONOS	**OpenDayLight**	**Poc SLEE Controller**
NorthBound APIs	Rest API	Rest API	SBB Interface
SouthBound APIs	OF 1.0, OF 1.3 NETCONF	OF 1.0, OF 1.3, NETCONF, SNMP, YANG	OF 1.0
Application Domain	Datacenter, WAN and Transport	DataCenter	Telecom, Carrier-Grade

2.7 Controller Comparison

Throughout the search for the base project, many SDN controllers have been found. Some focused on specific tasks and others on creating an environment for multipurpose. In this project only the carrier-grade SDN controllers such as ONOS (ONOSPROJECT, 2014) and OpenDayLight (MEDVED et al., 2014) in addition to the first version of CREDENCE (FERREIRA et al., 2014) have been chosen as a basis for testing and making criteria for creating the new *OpenFlow* controller.
For the comparison, some tests performed with Cbench (SHERWOOD; YAP, 2016) in another projects where it has been used. Cbench is a benchmarking tool for controllers, and in this case, for SDN controllers.
Basically the comparative is focused on a set of requirements to consider for the creation of a *Carrier Grade* controller capable of being implemented in a telecommunications environment and to have the support to work together with JAIN SLEE.
The table 1 compares some of the most relevant characteristics of the project, such as the *OpenFlow* protocol they handle, as well if it has options to implement some other modules using API.
Consider differences and similarities, the shortcomings of each controller were analyzed, focusing on the timeliness and ease of implementation in a telecommunications environment, and being able to integrate with other projects using JAIN SLEE.
It seems that the PoC SLEE Controller was the only one in the group capable of being easily implemented, and that its implementation does not involve so many processing costs inside the processor. That is why the choice to perform an update to the controller giving new libraries and possibilities, became a viable path and taking advantage of the characteristics of modulation.

Chapter 3

The CREDENCE Controller

In this chapter we disaggregate the term of CREDENCE to express a little the architecture that is used, as well as the main characteristics developed, including the Generic Translator in section 3.2, the Resource Adaptor in section 3.2.1 and the two SBB, TopologyGraph SBB in section 3.3.1 and Learning Switch SBB in section 3.3.2.

3.1 Architecture Overview

A controller created under the specifications of JAIN SLEE, using Java programming language and using Maven as software tool for the management and construction of projects.the controller was designed at first, using the basic rules of analysis and design of programming they were raised the requirements for that controller.
In principle only work as a translator, who will be responsible for sending/receiving packets of messages from external systems using as *OpenFlow* protocol and which in turn need to direct their messages to a higher entity (or lower depending on the case), for messages arriving from the controller will need to adapt the events that receive and decompress the instructions that will be transmitted to the south bound systems. Otherwise, each message sent by the lower layer, it will become an object later in the controller functionality, trigger an event in the application layer through the communication of OFRA (OpenFlow Resource Adaptor) with the help of *OpenFlow* Resource Adaptor Type.
The architecture was designed in the Entity Title Architecture (ETArch) who is an architecture for the title model with a focus on aggregating multicast traffic (GONÇALVES et al., 2014) plan for the clean-slate future internet, in cooperation with other research groups, which aimed to create a more robust architecture and allowing the use in demanding environments where their results incentivize the development of the region as show in the Figure 9.

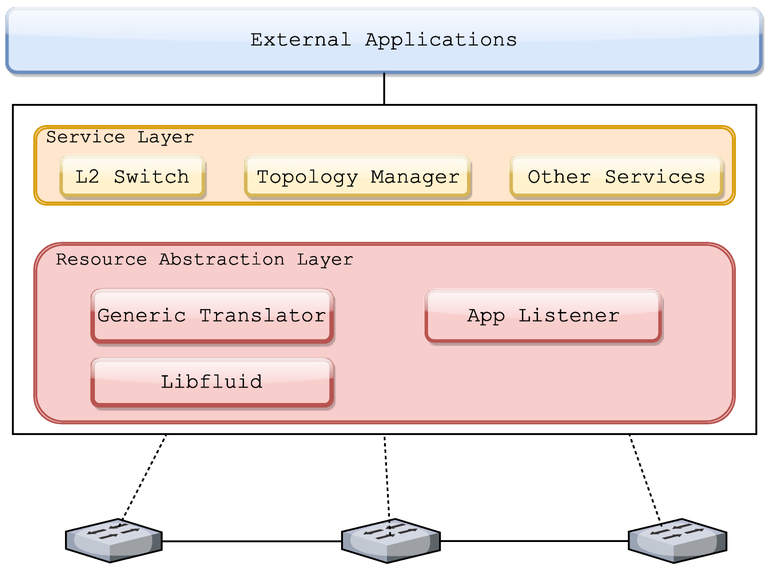

Figure 9 – Credence Architecture

3.2 Resource Abstraction Layer

Within the controller architecture is the implementation of the Libfluid library, which in the previous chapter was broken down a bit and commented in general terms what it meant. At this time, we use the core for the development of the project, recalling that initially, the library was conceived in c ++ but for communication with Jain Slee and other elements of the project is necessary that communication is developed in Java, it is why we focus on the use of the examples of Libfluid made in Java as well as its libraries, which were transformed using *Swig* (BEAZLEY et al., 1997) who is a software development tool that connects programs written in C and C ++ with a variety of high-level programming languages, in this case Java.
Another important point for the creation of Generic Translator is the use of libraries specifications messages *OpenFlow*, such as 1.0 and 1.3 than in previous ETArch projects have not yet been implemented.

3.2.1 Fault Tolerant Resource Adaptor (RA)

For the communication with the Mobicents Enviroment, it necessary that the adapter receives(and sends) the *OpenFlow* Messages within the controller and the mobicents solution system.
Since Mobicents is the first and only Open Source Jain Slee 1.1 certified product and *Jain Slee* is an application server (MOBICENTS, 2016), the resource adapter bridges the application model with the underlying event structure and can be designed in any language that is able to trigger events into the *JainSlee* application server .
For the project solution requires a Resource Adapter for the *OpenFlow* protocol besides

Table 2 – OfTypeMessages

	OpenFlow 1.0	**OpenFlow 1.3**	
		*** Immutable messages. ***	
0	OFPT_HELLO	OFPT_HELLO	Symmetric message
1	OFPT_ERROR	OFPT_ERROR	Symmetric message
2	OFPT_ECHO_REQUEST	OFPT_ECHO_REQUEST	Symmetric message
4	OFPT_ECHO_REPLY	OFPT_ECHO_REPLY	Symmetric message
	OFPT_VENDOR		
		*** Switch configuration messages. ***	
5		OFPT_EXPERIMENTER	Symmetric message
6	OFPT_FEATURES_REQUEST	OFPT_FEATURES_REQUEST	Controller /switch message
7	OFPT_FEATURES_REPLY	OFPT_FEATURES_REPLY	Controller /switch message
8	OFPT_GET_CONFIG_REQUEST	OFPT_GET_CONFIG_REQUEST	Controller /switch message
9	OFPT_GET_CONFIG_REPLY	OFPT_GET_CONFIG_REPLY	Controller /switch message
10	OFPT_SET_CONFIG	OFPT_SET_CONFIG	Controller /switch message
		*** Asynchronous messages. ***	
11	OFPT_PACKET_IN	OFPT_PACKET_IN	Async message
12	OFPT_FLOW_REMOVED	OFPT_FLOW_REMOVED	Async message
13	OFPT_PORT_STATUS	OFPT_PORT_STATUS	Async message
		*** Controller command messages. ***	
14	OFPT_PACKET_OUT	OFPT_PACKET_OUT	Controller /switch message
15	OFPT_FLOW_MOD	OFPT_FLOW_MOD	Controller /switch message
16		OFPT_GROUP_MOD	Controller /switch message
17	OFPT_PORT_MOD	OFPT_PORT_MOD	Controller /switch message
18		OFPT_TABLE_MOD	Controller /switch message
		*** Statistics messages. ***	
19	OFPT_STATS_REQUEST	OFPT_MULTIPART_REQUEST	Controller /switch message
20	OFPT_STATS_REPLY	OFPT_MULTIPART_REPLY	Controller /switch message
		*** Barrier messages. ***	
21	OFPT_BARRIER_REQUEST	OFPT_BARRIER_REQUEST	Controller /switch message
22	OFPT_BARRIER_REPLY	OFPT_BARRIER_REPLY	Controller /switch message
		*** Queue Configuration messages. ***	
23	OFPT_QUEUE_GET_CONFIG_REQUEST	OFPT_QUEUE_GET_CONFIG_REQUEST	Controller /switch message
24	OFPT_QUEUE_GET_CONFIG_REPLY	OFPT_QUEUE_GET_CONFIG_REPLY	Controller /switch message
25		OFPT_ROLE_REQUEST	Controller /switch message
26		OFPT_ROLE_REPLY	Controller /switch message
27		OFPT_GET_ASYNC_REQUEST	Controller /switch message
28		OFPT_GET_ASYNC_REPLY	Controller /switch message
29		OFPT_SET_ASYNC	Controller /switch message
30		OFPT_METER_MOD	Controller /switch message

offering non-functional basic requirements, support fault tolerance, since the focus of the controller is a vision of *Carrier Grade.*
The resource adapter will be responsible for interaction with *JainSlee.* Where it will be the recipient of messages from external systems using the *OpenFlow* protocol, in this case using the AppListener as receiver and Generic translator as encoder and sent as events that are produced within the Resource Adaptor for SBB.

In the AppListener class, the handling of OFTypeMessages is included for both the *OpenFlow* 1.0 protocol and the *OpenFlow* 1.3 protocol as described in the Table 2. So the Resource Adaptor will consume events that are sent by the service within *JainSlee* and transmits to all OpenFlow-enabled switches.

In the table can appreciate the different types of *OpenFlow* Messages that do currently part of the specifications of both *OpenFlow* 1.0 as *OpenFlow* 1.3, so, in the elaboration of the controller took into account all message types to encapsulate them with the help

of *libfluid_msg* for objects of their type of message routing and also for the upper layers, leaving the processing controller, and to offer the possibility of implementing measures in *Carrier Grade* management messages. The *JainSlee* development includes both RA Resource Adapter Type, the Resource Adapter and the Fault Tolerant Mode. Starting with the Resource Adapter which includes the *OpenFlow* Protocol Stack which involves the implementation of the libfluid library discussed in the section 2.5 and implement a listener able to attend the events that are fired from the previous class as well as the responses are sent from the results of the blocks or SBB's service.
The Resource Adapter also implements an extern class called *OpenFlow* Resource Adapter Type where all events of_Messsage were declared within the XML for both versions *OpenFlow* 1.0 to *OpenFlow* 1.3 versions, also specifies the events that will be launched with the creation of objects thanks to the class of OpenFlow.events.OFP_TYPE and converted to events for the application layer like the Figure 8.
Already entered in the field, with the question of implementing a Fault Tolerance mode which is a support that *JainSlee* offer for the information clustering, giving to the system the opportunity to create a module for High Availability(HA) or Fault Tolerance (FT) support.

> The fault tolerant mode is a fully clustered mode with state replication. An FT cluster can be viewed as one virtual container that extend over all the *JainSlee* nodes that are active in the cluster. All activity context and SBB entity data is replicated across the cluster nodes and is hence fully redundant. Events are not failed over, due to performance constraints, which means that an event fired and not yet routed will be lost if its cluster node fails (MOBICENTS, 2016).

For the implementation of Fault Tolerance Resource Adapter is required the implementation of an Application Programming Interface (API), which is called Fault Tolerance Resource Adapter API, which is included in the version 1.1 of *JainSlee* and extends the *JainSlee* 1.1 Resource Adapter API providing missing features related to clustering. An effort has been made keep the API similar to the standard RA contract, so that anyone who has developed a *JainSlee* 1.1 RA is able to easily use the proprietary API extension. Its implementation is developed in the core of the RA, since it is necessary to trap errors and replicate data to provide greater capacity to the system to recover data in one way or another they have been involved in the transmission and have been damaged.

3.2.2 Generic Translator

In the Resource Abstraction Layer we found the Generic Translator who is responsible for receiving all messages from the switch and host, which will have to be parsed in the first instance, this call south bond, because it is in that layer where the packets are coming. The process begins with the reception of messages, acquired by a AppListener class created in order to create the communication socket with the Resource Adapter and listen to the port that has been previously and according to the new rules of *OpenFlow* set, this default port 6653, but made the clarification that it is possible to make the change. Taking up the message path, and received by the AppListener, this is sent to the Generic Translator, a Java class that includes the libraries of libfluid_msg, where besides being able to differentiate the specification of Open Flow of the message, will create objects of type message reaching the controller.
This being the way, the Generic Translator is designed for create and fire objects that inherit the type of message that arrives and its specification, which will then be released to be converted into an event who on *JainSlee* can be of 2 types, can be an event consumer or an event producer who are part of the SLEE architecture and are who control the flow of the message either from the controller to the switch or switch to the controller.
Another function of the Generic Translator is to take the instruction that were send by the Service Layer and pass by the App Listener, to be converted into a flow mod to send it to the SouthBound layer, that include switches and another network elements.

3.3 Service Layer

The Service Layer contains the behavior of the CREDENCE SDN Controller. Two basic services were created: one that is responsible to handle the network graph (Topology Manager) and another one that implements a L2 Learning Switch. Each event that is triggered by the RA is received by one or more SBB's, that are responsible for processing the event and, if so, respond to the lower layers. This is where there are better opportunities for modularity, since SBB's can be created individually and only need to be called from the specifications in the RA. Using the *JainSlee* component model it is possible to create different SBBs in order to integrate the SDN Controller with other telecommunication services already provided by SBBs deployed at the operator.

3.3.1 TopologyGraph Service Building Block (SBB)

To demostrate the controller operation, it is necessary to create a service that uses its capabilities. For it has made the construction of a SBB for creating a graph of the network, or in other words a graph of the network topology.

A network topology is defined as the representation of how the network is designed, whether on a logical or physical plane. The topology is formed by a set of interconnected nodes and depends on the architecture of the network, the type of topology that it handles.
A topology administrator checks the state of the connections as well as who stores the basic data of their hosts, in this case they can be switches, controllers, hosts and the connection links between all the nodes.
Currently in the connection between the controller and the switch *OpenFlow* 1.0 is transmitted through a OFPT_FEATURE_REQUEST and a OFPT_FEATURE_REPLY, which is a *OpenFlow* data type that includes the MAC addresses of each switch as well as the ID's of the ports, to be stored in a data table on the controller, this is basically how to make the discovery of all nodes connected to the network.
Different from the operation in the *OpenFlow* 1.3 specification, since the header that is sent by the controller is now an OFPT_MULTIPART_REQUEST data type in which the option of the controller is added in the body of message, The following table lists the names of the objects in the list: GroupName, GroupDesc, GroupFeatures, Meter, MeterFeatures, TableFeatures, PortDesc, Experimenter like the Figure 10. And it is the switch that responds with a OFPT_MULTIPART_REPLY the request of the controller about the information that is needed.
For the case of SBB, the description information is requested, which includes the basic

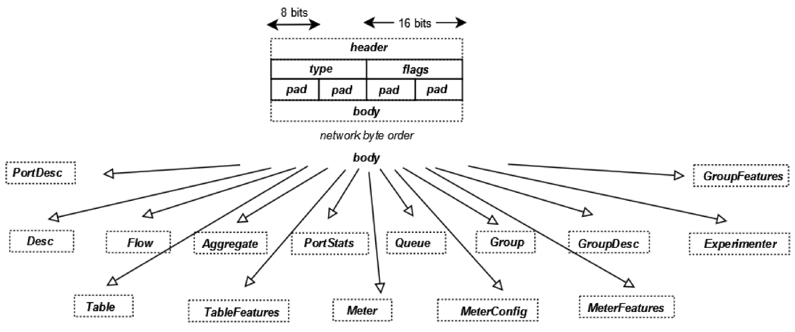

Figure 10 – MultipartRes Structure (FLOWGRAMMABLE, 2016)

information of the switch and also the PortDesc value is requested to obtain the information of its ports (as in the protocol *OpenFlow* 1.0).
The library in C++ has all classes defined for objects and the object vectors that have each of their answers, something different happens in Java because when using SWIG (BEAZLEY et al., 1997) in the original files that the Libfluid Team developed, they did not include the support for those vector objects included within the message, so a new compilation was made adding the elements that would make possible the use of the content of the vectors that arrive through the messages *OpenFlow*.
With all the information already in the SBB module, it can be requested by the Northbound layer with the most prevalent choice in the last days. APIs in Software Define

Network(SDN) (ZHOU et al., 2014) or some other SBB that needs the information collected there, the messages are exported through a logger but can be altered according to the needs.
The basic difference between the specifications are the upgrades that over time has had the specification, since it allows to obtain more data and in turn a better handling of the information that the switch supplies.

3.3.2 LearningSwitch Service Building Block (SBB)

The L2 is responsible for receiving *OpenFlow* 1.0 and 1.3 message packets and examine each packet to learn the sourceport mapping. So, its creates the flow table including the Source and destination MAC address when the packet arrive, if is not has been associated, the packet will be flooded on all ports of the switch, that work only once per set.
Once the flowMod has been created, the switch can send the packet to the given port
In other controllers, who processes and create the flowMods is the controller itself as shown in the Figure 11.

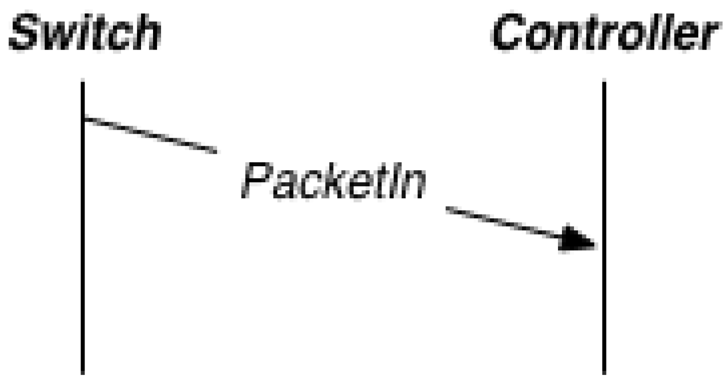

Figure 11 – Packet in (FLOWGRAMMABLE.ORG, 2016)

The case changes when an *JainSlee* implementation with a controller focused on *Carrier Grade* thus avoid job processing in the controller layer and tasks to the top layer is transferred.
Within the upper layer a service building block (or SBB) is created, which is responsible for processing the messages, creating the flow table and routing the messages to the given port, thus basically releasing a significant load for the controller.
In the top layer a block of service (or SBB), which is responsible for processing messages, create the flow table and route the messages to the given port, so basically a significant burden on the controller is released is created.
Bringing in a long range will allow focus on providing services Connection with the low layer and the SLEE ecosystem for so the ability to manage the flow of the message.

Since the L2 Learning Switch SBB is only for routing and creation of *OpenFlow* message packet tables, SBB's capability goes beyond that and provides options for triggering one or more activities (if necessary) for each one of them.
One thing to take into account in the analysis process for future deployments happens within the parsing of *OpenFlow Message*:
This is due to the difference of specifications that are handled inside the controller, either for *OpenFlow* 1.0 and for *OpenFlow* 1.3. Since from specification 1.2, the message structure of Packet_in was altered to include other elements, from the initial structure (Figure 12) to receive an update and finish in the structure used by specification 1.3 (Figure 13) can be noted that the value in_port was deleted to make space for other variables, modifying the operation of SBB, since when it receives a Packet_In, expects to find the value of the variable in_port.
For this eventuality it became necessary to obtain the value of the variable from "Match" in its getValue method.

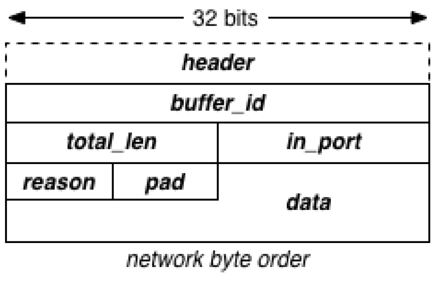

Figure 12 – Structure of packet_in 1.0 (FLOWGRAMMABLE.ORG, 2016)

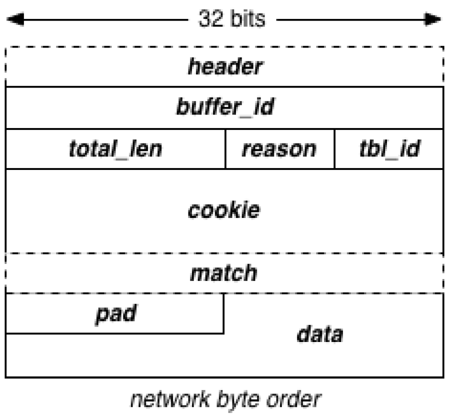

Figure 13 – Structure of packet_in 1.3 (FLOWGRAMMABLE.ORG, 2016)

3.4 The NorthBound Interface

Thanks to the benefits of implementing SDN is possible to have a capable communication layer between the SDN controller and the services and applications running over the network. The North Bound Interface (NBI) is oriented to *Carrier Grade* networks and telecommunications networks where it is able to implement some services.
Several of the tools that are used daily by telecommunications and related companies, Involve low-level data modification to be able to give a better result of their actions such as routing or load balancing, firewall services, FTP connection and many other services that are accessed through controllers or platforms that offer the option to modify from the upper layers, values and actions of the lower layers without having to specifically alter individual rules at the machine level.
The services are usually implemented with the following interfaces:

- ❏ Extensible Markup Language (XML)
- ❏ File Transfer Protocol (FTP)
- ❏ Simple Network Management Protocol (SNMP)
- ❏ System Log (SYSLOG)
- ❏ Terminal Access Controller Access Control System (TACACS)
- ❏ Representational State Transfer (REST)
- ❏ Lightweight Directory Access Protocol (LDAP)

This allows endless probabilities due to the ability to create new SBB with specific functionalities and that also can be integrated with the controller.
Orchestration platforms such as OpenStack (OpenStack Foundation, 2013) can be integrated with the SDN controller and the SDN NorthBound APIs Thus creating a disruption in the operation of the network where it is the developers are able to make changes in the lower layers of communication without having to be directly involved in the adaptations.

Chapter 4

CREDENCE CONTROLLER Deploy and Validation

In the validation chapter, the tests carried out by the LibFluid team to demonstrate the main functionalities of the library have been taken as a basis, thanks to the help of Collective Benchmark (cBench) (SHERWOOD; YAP, 2011). Cbench is a benchmarking tool for *OpenFlow* controllers. The tool pretends to be a variable number of switches and each switch establish a session with the benchmarked controller. CBench is a common tool used by several other studies in the literature (SALMAN et al., 2016) (ZHAO; IANNONE; RIGUIDEL, 2015) (ANDRADE et al., 2016). CBench act as a traffic generator sending *OpenFlow* a *Packet In* (OFPT_PACKET_IN) (FOUNDATION, 2012) as fast as possible and receives *Flow Mod* (OFPT_FLOW_MOD) (FOUNDATION, 2012) messages.

4.1 Testbed Description

The base tests were carried out with the creation of 2 SBB's, since they were necessary to check the connections and consume the services of the Resource Adapter.
Using the L2 learning switch documented in the section 3.3.2, the tests were performed and under the same conditions, the other 2 controllers were validated. For the first two tests a comparison was made with the results obtained on the controllers and for the last test, the validation was performed only with the CREDENCE controller.
The benchmarks were performed on a VM equipped with an Intel Core i7-4400MQ CPU (4 cores, 4 threads @ 2.4 GHz) with 3 GB of RAM (@1666 MHz) running Fedora 20 and all the Jain Slee Stack including Eclipse Luna, Java and its JDK 1.8.0.
For ONOS we use the 1.3.0 Version with only the forwarding active plugin and for OpenDayLight we use the version 1.8.0 which was the last one presented in December of 2016.

4.2 Experiments

In this chapter we present the experiments performed as the results obtained on the controller using the CBENCH test tool and in the annex 1 the instructions for setting up the environments of OpenDayLight, Onos and CREDENCE controller.

The command for CBench has the next configuration options
Where:
-c controller (IP or hostname)
-p controller port number
-m test time per s
-l loops per test
-s # of switches
-M #of mac addresses per switch
-t throughput mode

4.2.1 Test 1: Controller throughput

This test shows a raw measurement of the throughput in different controllers and compares them to the Credence controller. We have to keep in mind that the learning switch implementation includes several differences on its application for each controller. This experiment was conducted using the following parameters:

Switches: 8

Controllers: Credence, ONOS, OpenDayLight

Threads: 4

Application: L2 Learning switch

Command: *cbench -c localhost -p 6653 -m 10000 -l 4 -s 8 -M 1000000 -t*

Results (higher is better):

Table 3 – Controller Throughput

Controller	**flows/ms**
ONOS	3170.56
OpenDayLight	3584.89
Credence	3594.00

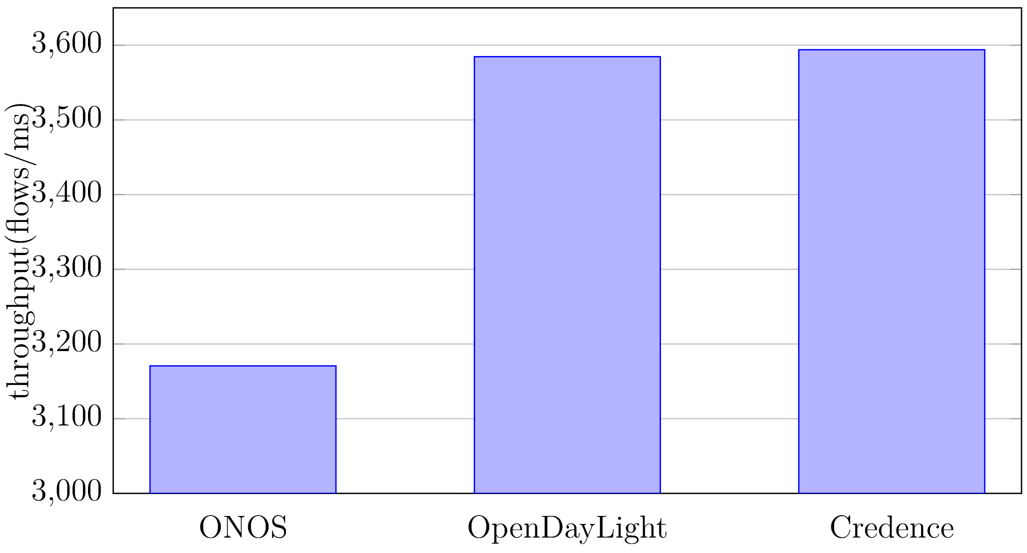

Figure 14 – Controller Throughput

4.2.2 Test 2: Controller latency

Description: This test shows a comparison of the latencies introduced by different controllers running a learning switch application and compares them to libfluid. These values show just part of the picture, since there are differences in the implementation of the learning switch in different controllers.This experiment was conducted using the following parameters:
Switches: 16
Controllers: Credence, ONOS, OpenDayLight
Threads: 8
Application: learning switch
Command: *cbench -c localhost -p 6653 -m 10000 -l 16 -s 16 -M 1000000*
Results (lower is better):

Table 4 – Controller Latency

Controller	**ms**
ONOS	0.028562
OpenDayLight	0.0283140
Credence	0,0250765

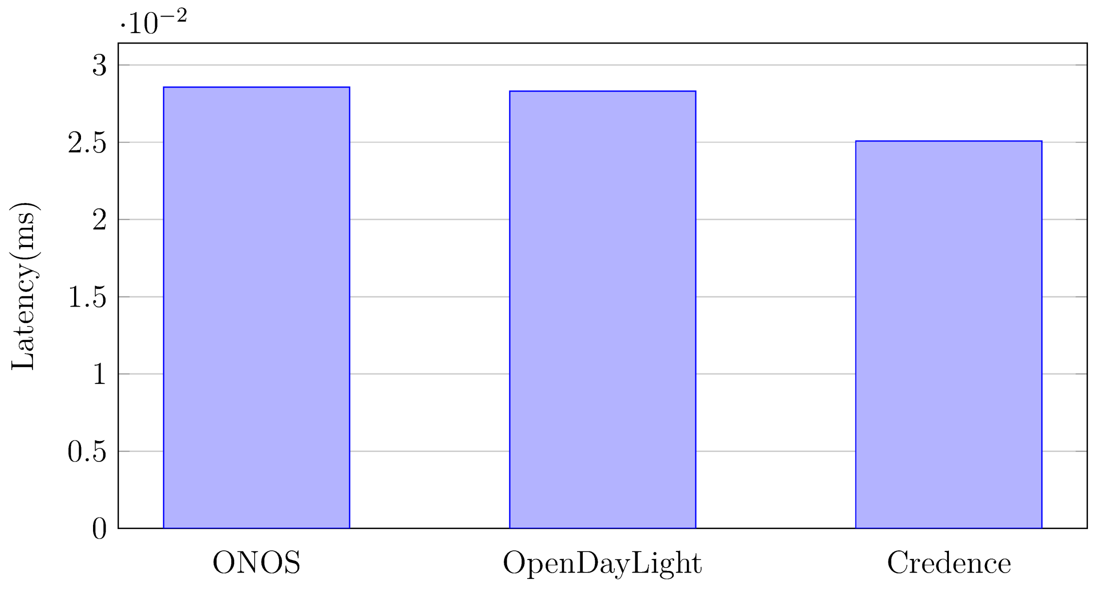

Figure 15 – Controller Latency

4.2.3 Test 3: Network size performance

Description: This test shows how CREDENCE performance scales when increasing the network size. The number of MACs per switch was reduced (from 10000k to 1000k) to avoid thrashing.This experiment was conducted using the following parameters:

Switches: 1, 4, 16, 64, 128, 256,512

Controllers: Credence

Threads: 8

Application: learning switch

Command: *cbench -c localhost -p 6653 -m 10000 -l 16 -s 16 -M 1000000 -t* 16

Table 5 – Network Size Performance

Switches	**flows/ms**
1	580.12
2	3564.79
4	4394.16
16	3931.53
64	4561.84
128	4518.11
256	3323.65
512	560.05

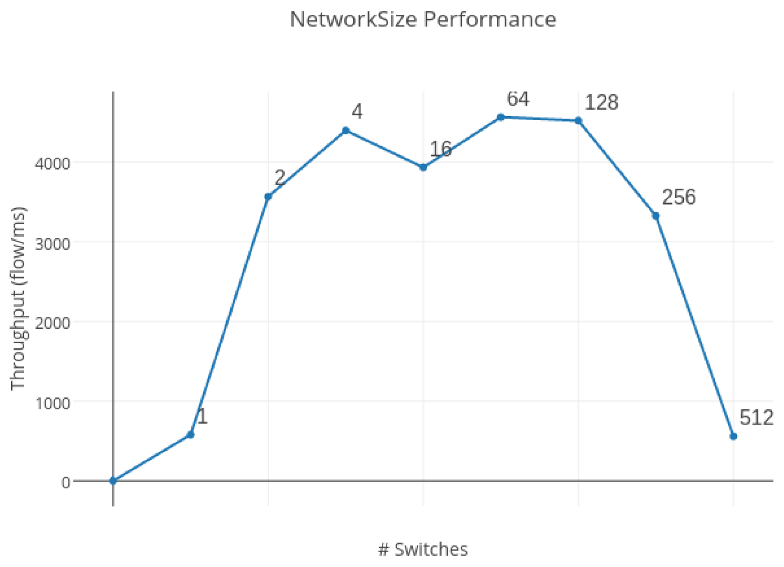

Figure 16 – Network Size Performance

4.3 Comparative Analysis

Given the results obtained (Table 6), we can determine that the CREDENCE controller had a good performance in the comparative tests, this can be determined thanks to the controller's livery and its functionalities. Although the results in question of trought-put were very disputed with the controller OpenDayLight and in question of latency they were very similar. In the test 3 (Table 7) we also note the ability of the controller to operate in a large network and that taking advantage of the Carrier Grade environment offered by Jain Slee, is able to provide a large scale support with good response times. It leaves clear to see the opportunity that *CREDENCE* has to become a very good option of implementation with architectures like *ETArch* without its performance is seen Affected to a large extent and on the contrary a much greater opportunity for implementation.

Table 6 – Raw Test 1 and 2

Controller	***Throughput (flows/ms)***				**Latency (flows/ms)**				
	Min	**Max**	**Avg**	**Stdev**	**Min**	**Max**	**Avg**	**Stdev**	**ms**
ONOS	3103.68	3237.45	3170.56	66.88	2621.39	2856.20	2738.80	117.40	0.028562
Credence	2995.21	4192.79	3594.00	598.79	2491.90	2831.40	2507.65	15.75	0.0250765
OpenDayLight	1757.57	4812.21	3584.89	1827.32	2831.40	2523.39	2507.65	0.0	0.0283140

Table 7 – Raw Test 3

Switches	**Min(flows/ms)**	**Max(flows/ms)**	**Avg(flows/ms)**	**Stdev(flows/ms)**
1	536.47	623.77	580.12	46.65
2	2754.15	4375.44	3564.79	810.65
4	4394.16	4394.16	4394.16	0.00
16	3931.53	3931.53	3931.53	0.00
64	4561.84	4561.84	4561.84	0.00
128	4518.11	4518.11	4518.11	0.00
256	3323.65	3323.65	3323.65	0.00
512	503.90	616.20	560.05	56.15

Chapter 5

Conclusion

In this paper, we have presented the CREDENCE SDN controller. The initial evaluation results are very encouraging, indicating that the CREDENCE controller has the lowest latency when compared to ONOS, OpenDayLight (ODL), Ryu and POX. Also, the CREDENCE controller presented the best throughput when compared to the same controllers.

The results shows that using the CREDENCE controller it will be possible to develop solutions based on the *JAIN SLEE* platform that can integrate SDN with telecom services capable of offering better performance and possibilities of innovation in the field of network. By Using the CREDENCE controller the operator does not need to deploy a new entity in the network but can integrate with the JAIN SLEE already deployed, resulting in lower CAPEX.

Credence includes in its core LibFluid library with which the objectives were achieved to create a lightweight, fast and functional controller that would allow to work on the according to the required requirements. The system currently supports the creation of SBB or also called events, which can consume the capabilities of the controller to receive and send the messages and only requires the declaration of events in the configuration file of the Resource Adapter.

5.1 Future Work

As future work we will carry out a in-depth study aiming to show all CREDENCE Controller's carrier grade capabilities compared to the other evaluated controllers. Another possible work is the integration of the SBB's of the ETArch architecture into this new version of CREDENCE Controller, create an SBB that would allow you to create a Northboud API that could be used so that other applications could communicate with the CREDENCE CONTROLLER and create an abstraction similar to ONOS INTENT.

Bibliography

ANDRADE, L. et al. On the Benchmarking Mainstream Open Software-Defined Networking Controllers. In: **Proceedings of the 9th Latin America Networking Conference**. New York, NY, USA: ACM, 2016. (LANC '16), p. 9–12. ISBN 978-1-4503-4591-0. Disponível em: <http://doi.acm.org/10.1145/2998373.2998447>.

BEAZLEY, D. et al. Swig users manual (version 1.1). University of Utah, 1997.

BERDE, P. et al. Onos: Towards an open, distributed sdn os. In: **Proceedings of the Third Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 1–6. ISBN 978-1-4503-2989-7. Disponível em: <http://doi.acm.org/10.1145/2620728.2620744>.

CHIOSI DON CLARKE, P. W. A. R. M. Network functions virtualization – introductory white paper. **SDN and OpenFlow World Congress**, oct 2012.

DAS, S.; PARULKAR, G.; McKeown, N. Why OpenFlow/SDN can succeed where GMPLS failed. In: **European Conference and Exhibition on Optical Communication**. [S.l.]: Optical Society of America, 2012. p. Tu–1.

DIXIT, A. et al. Towards an elastic distributed sdn controller. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 43, n. 4, p. 7–12, ago. 2013. ISSN 0146-4833. Accessed: 2017-01-30. Disponível em: <http://doi.acm.org/10.1145/2534169.2491193>.

FEMMINELLA, M. et al. Scalability and performance evaluation of a JAIN SLEE-based platform for VoIP services. In: **Teletraffic Congress, 2009. ITC 21 2009. 21st International**. [S.l.: s.n.], 2009. p. 1–8.

Femminella, M. et al. Implementation and performance analysis of advanced IT services based on open source JAIN SLEE. In: **2011 IEEE 36th Conference on Local Computer Networks (LCN)**. [S.l.: s.n.], 2011. p. 746–753.

FERREIRA, C. et al. Towards a carrier grade SDN controller: Integrating OpenFlow with telecom services. In: . [s.n.], 2014. p. 70–75. ISBN 978-1-61208-360-5. Disponível em: <http://www.thinkmind.org/index.php?view=article&articleid=aict_2014_3_40_10162>.

FERRY, D. **JAIN SLEE (JSLEE) 1.1 Specification, Final Release**. 2. ed. Oracle, 2008. Disponível em: <http://download.oracle.com/otn-pub/jcp/jain_slee-1_1-final-oth-JSpec/jslee-1_1-fr-spec.pdf?AuthParam=1434654213_688966ba47d0e44016a596058f8490b1>.

FLOWGRAMMABLE. **MultipartReply structure SDN OpenFlow Message Layer PacketIn**. 2016. <http://flowgrammable.org/sdn/openflow/message-layer/statsreponse/>. Accessed: 2016-11-30.

FLOWGRAMMABLE.ORG. **PacketIn SDN OpenFlow Message Layer PacketIn**. 2016. <http://flowgrammable.org/sdn/openflow/message-layer/packetin/>. Accessed: 2016-11-30.

FOUNDATION, O. N. **OpenFlow Switch Specification. Version 1.3.0**. 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf>.

GONÇALVES, M. A. et al. Etarch: projeto e desenvolvimento de uma arquitetura para o modelo de título com foco na agregação de tráfego multicast. Universidade Federal de Uberlândia, 2014.

GOOGLE. **A scalable control platform writen in Java which supports OpenFlow switches**. 2015. Disponível em: <http://code.google.com/p/maestro-platform>.

HAN, S.; LEE, S. Persistent store-based dual replication system for distributed SDN controller. In: **2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)**. [S.l.: s.n.], 2016. p. 1–2.

Intel Embedded Community. **Don’t Confuse “High Availability” with “Carrier Grade”**. 2014. Disponível em: <https://embedded.communities.intel.com/community/en/blog/2014/04/04/don-t-confuse-high-availability-with-carrier-grade>.

KOPONEN, T. et al. Onix: A distributed control platform for large-scale production networks. In: **OSDI**. [s.n.], 2010. v. 10, p. 1–6. Accessed: 2015-09-24. Disponível em: <http://static.usenix.org/events/osdi10/tech/full_papers/Koponen.pdf>.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

MARTINO, C. D.; MENDIRATTA, V.; THOTTAN, M. Resiliency Challenges in Accelerating Carrier-Grade Networks with SDN. In: **2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)**. [S.l.: s.n.], 2016. p. 242–245.

McCauley, J. et al. Extending SDN to large-scale networks. p. 1–2, 2013. Accessed: 2015-09-24. Disponível em: <http://www.cs.columbia.edu/~lierranli/coms6998-8SDNFall2013/papers/Xbar-ONS2013.pdf>.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Comput. Commun. Rev.**, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Accessed: 2015-09-24. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MEDVED, J. et al. Opendaylight: Towards a model-driven sdn controller architecture. In: **2014 IEEE 15th International Symposium on**. [S.l.]: IEEE, 2014. p. 1–6.

MIJUMBI, R. et al. Network Function Virtualization: State-of-the-Art and Research Challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, 2016. ISSN 1553-877X.

MOBICENTS. **Chapter6.Mobicents JAIN SLEE Clustering**. 2016. Accessed: 2016-11-24. Disponível em: <https://docs.jboss.org/mobicents/jain-slee/2.7.0.FINAL/container/user-guide/en-US/html/clustering.html>.

MOBICENTS.ORG. **Mobicents**. 2016. Disponível em: <http://www.mobicents.org/>.

MONCLUS, P. **Control Plane - Data Plane Separation: Architecture Gridlock| The Worldwide Leader of Secure Cloud Networks**. 2013. Accessed: 2016-11-30. Disponível em: <http://www.plumgrid.com/plumgrid-blog/2013/03/control-plane-data-plane-separation-architecture-gridlock/>.

NICIRA. **NOX, The original OpenFlow controller for development of fast C++ controllers on Linux.** 2011. Disponível em: <http://www.noxrepo.org/nox/about-nox/>.

ON.LAB. **Open Network Operating System v2**. 2013. Disponível em: <http://onosproject.org/software/#releases>.

ONOSPROJECT. **Whitepaper ONOS-final**. 2014. Disponível em: <http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf>.

OPENDAYLIGHT. **ODL Beryllium (Be) - The Fourth Release of OpenDaylight | OpenDaylight**. 2016. Disponível em: <https://www.opendaylight.org/odlbe>.

OPENFLOW.STANFORD.EDU. **Beacon, Fast OpenFlow Controller**. 2015. Disponível em: <https://openflow.stanford.edu/display/Beacon/Home>.

OpenStack Foundation. **OpenStack networking administration guide**. 2013. <http://http://docs.openstack.org/index.html>. [Online; accessed 19-Dec-2016].

PELTZ, C. Web services orchestration and choreography. **Computer**, IEEE Computer Society, Los Alamitos, CA, USA, v. 36, n. 10, p. 46–52, 2003. ISSN 0018-9162. Accessed: 2016-11-30.

PITT SEDEF OZCANA, G. P. D. Ons open networking summit. In: **ONS is the Established Premier SDN & NFV Conference**. [S.l.: s.n.], 2014. p. –.

RCR Wireless. **What makes ‘carrier-grade’ reliability so hard?,2015**. 2015. Accessed: 2015-12-12. Disponível em: <http://www.rcrwireless.com/20140512/opinion/reader-forum-makes-carrier-grade-reliability-hard>.

RYU. 2011. Accessed: 2017-02-23. Disponível em: <https://ryu.readthedocs.io/en/latest/getting_started.html#what-s-ryu>.

SALMAN, O. et al. SDN controllers: A comparative study. In: . IEEE, 2016. p. 1–6. ISBN 978-1-5090-0058-6. Accessed: 2016-11-30. Disponível em: <http://ieeexplore.ieee.org/document/7495430/>.

SDNCENTRAL. **The Future of Network Virtualization and SDN Controllers**. Sunnyvale, CA, 2016. 52 p. Accessed: 2015-09-24. Disponível em: <https://www.sdxcentral.com/reports/network-virtualization-sdn-controllers-download-2016/>.

SDXCENTRAL. **Inside SDN - Architecture**. 2015. Accessed: 2016-11-30. Disponível em: <https://www.sdxcentral.com/resources/sdn/inside-sdn-architecture/>.

SHALIMOV, A. et al. Advanced study of SDN/OpenFlow controllers. In: **Proceedings of the 9th Central &#38; Eastern European Software Engineering Conference in Russia**. ACM, 2014. (CEE-SECR '13), p. 1:1–1:6. ISBN 978-1-4503-2641-4. Accessed: 2016-11-30. Disponível em: <http://doi.acm.org/10.1145/2556610.2556621>.

SHERWOOD, R.; YAP, K. Cbench controller benchmarker. **Last accessed, Nov**, 2016.

SHERWOOD, R.; YAP, K.-K. **Cbench Controller Benchmarker**. 2011. Disponível em: <https://github.com/mininet/oflops/tree/master/cbench>.

SILVA, F. et al. Enabling a carrier grade sdn by using a top-down approach. **Anais do V Workshop de Pesquisa Experimental da Internet do Futuro - WPEIF 2014 25**, 2014. Disponível em: <http://www.sbrc2014.ufsc.br/anais/files/wpeif/ST2-5.pdf>.

SUH, D. et al. On performance of OpenDaylight clustering. In: **2016 IEEE NetSoft Conference and Workshops (NetSoft)**. [S.l.: s.n.], 2016. p. 407–410.

Vidal, A.; ROTHENBERG, C. E.; VERDI, F. L. The libfluid OpenFlow driver implementation. In: **32nd Brazilian Symposium on Computer Networks (SBRC). SBC**. [s.n.], 2014. p. 8. Accessed: 2016-11-30. Disponível em: <http://sbrc2014.ufsc.br/anais/files/salao/SF-ST3-2.pdf>.

VIDAL, A.; ROTHENBERG, C. E.; VERDI, F. L. The libfluid OpenFlow driver implementation. In: **32nd Brazilian Symposium on Computer Networks (SBRC). SBC**. [s.n.], 2014. p. 8. Accessed: 2016-11-30. Disponível em: <http://sbrc2014.ufsc.br/anais/files/salao/SF-ST3-2.pdf>.

WARWICK, M. **Telco spending on SDN and NFV forecast to reach US$157 billion by 2020**. 2015. Accessed: 2015-09-24. Disponível em: <http://www.telecomtv.com/articles/sdn/telco-spending-on-sdn-and-nfv-forecast-to-reach-us-157-billion-by-2020-12859/>.

YUE, F. Network functions virtualization—everything old is new again. **F5 Networks**, 2013.

ZHAO, Y.; IANNONE, L.; RIGUIDEL, M. On the performance of SDN controllers: A reality check. In: **2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)**. [S.l.: s.n.], 2015. p. 79–85.

ZHOU, W. et al. REST API Design Patterns for SDN Northbound API. In: **2014 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)**. [S.l.: s.n.], 2014. p. 358–365.

Appendix

APPENDIX **A**

Instalation Guide

Below are the guides for installing the environments in which the tests were developed, including the installation of the main requirements including JAVA and the JDK.

A.1 ONOS

Install curl:

```
#sudo yum instal curl
```

Install git :

```
#sudo yum install git
```

Install java 8:

```
#yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

Create Applications folder

```
#cd /home/etarch
#mkdir Applications
```

Download Maven and Apache Karaf:

```
#wget https://archive.apache.org/dist/karaf/3.0.5/apache-karaf-3.0.5.tar.gz
#wget https://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
```

```
#tar -zxvf apache-karaf-3.0.5.tar.gz -C ../Applications/
#tar -zxvf apache-maven-3.3.9-bin.tar.gz -C ../Applications/
```

Git Clone ONOS

```
    #git clone https://gerrit.onosproject.org/onos -b 1.3.0
```

Modify "bash_profile" file (With Maven and Karaf path)

```
    #/home/onos/tools/dev/bash_profile
#source ./tools/dev/bash_profile
#echo $KARAF_ROOT
```

Compile ONOS

```
    #cd /home/etarch/onos
#mvn clean install -DskipTests
#ok clean
```

Activate forwarding to use in cbench

```
    onos>cfg set org.onosproject.fwd.ReactiveForwarding packetOutOnly true
```

Make Cbench Test

A.2 OpenDayLight

Figures can be created directly in LaTeX, as shown in ??. We need Maven to build ODL. Install the most recent version of Maven

```
    # sudo mkdir -p /usr/local/apache-maven
```

First you have to download the maven source code

```
    #wget http://ftp.unicamp.br/pub/apache/maven/maven-3/3.3.9/binaries/apache-
maven-3.3.9-bin.tar.gz
```

Now install maven

```
# sudo mkdir /usr/local/apache-maven
# sudo mv apache-maven-3.3.9-bin.tar.gz /usr/local/apache-maven
# sudo tar -xzvf /usr/local/apache-maven/apache-maven-3.3.9-bin.tar.gz -C
/usr/local/apache-maven/
# sudo update-alternatives –install /usr/bin/mvn mvn /usr/local/apache-maven/apache-
maven-3.3.9/bin/mvn 1
# sudo update-alternatives –config mvn
```

select option 1

```
# sudo yum install vim
# vim /.bashrc
```

Add these to your /.bashrc
Very important to put the "m" on the end

```
export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.9
export MAVEN_OPTS="-Xms256m -Xmx512m"
```

Only if you doesnt' have install java yet

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk # This matches sudo update-
alternatives –config java
```

Use the /.bashrc

```
#source /.bashrc ifconfig
```

Now use git to build ODL

```
# sudo yum install git
# git clone https://git.opendaylight.org/gerrit/integration/distribution
```

Lesson Learned: You will get a "Parent no found" error unless you have the right settings.xml in your /.m2 folder

```
# curl https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml
–create-dirs -o  /.m2/settings.xml
# cd distribution
# mvn clean install -DskipTests
```

If you have mvn compile error, try install java 8.

```
#sudo yum install java-1.8.0-openjdk
#sudo yum install java-1.8.0-openjdk-devel
```

The java-1.8.0-openjdk package contains just the Java Runtime Environment. If you want to develop Java programs then install the java-1.8.0-openjdk-devel package.

```
#sudo update-alternatives –config java
```

Run Karaf (Takes about 3 minutes to start)

```
# cd distributions/karaf/target/assembly/bin
# ./karaf -of13
```

If it hangs, run

```
# ./karaf clean -of13
```

Be sure to restart ovs so you can use mininet

```
# sudo service openvswitch-switch start
```

Use integration project.

```
#wget https://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/org-
/opendaylight/integration/distribution-karaf/0.6.0-SNAPSHOT
#cd <extracted directory>/bin
#./karaf
```

or clone and build integration project:

```
    #git clone https://git.opendaylight.org/gerrit/p/integration.git
#cd integration/distributions/extra/karaf
#mvn clean install
#cd ./target/assembly/bin
```

and finally run

```
    #./karaf
```

If you want to use RESTCONF with openflowplugin project, you have to install odl-restconf feature to enable that. To install odl-restconf feature run the following command

```
    karaf#>feature:install odl-restconf
```

Enable openflowplugin rest

```
    karaf#>feature:install odl-openflowplugin-flow-services-rest
```

use controller/distribution/opendaylight project
download and unzip latest build from:
https://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/org/opendaylight/contr
karaf/1.8.0-SNAPSHOT/distribution.opendaylight-karaf-1.8.0-20161204.062753-162.zip

```
    cd opendaylight
```

or clone and build controller project:

```
    #git clone https://git.opendaylight.org/gerrit/p/controller.git
#cd home/distribution/distribution-karaf-0.6.0-SNAPSHOT/controller/opendaylight
mvn clean install
cd target/distribution.opendaylight-osgipackage/opendaylight
```

Install features and activate drop the packets

```
    opendaylight-user@root>feature:install odl-mdsal-common
opendaylight-user@root>feature:install odl-config-api
opendaylight-user@root>feature:install odl-config-netty-config-api
opendaylight-user@root>feature:install odl-config-core
opendaylight-user@root>feature:install odl-config-manager
opendaylight-user@root>feature:install odl-config-netty
opendaylight-user@root>feature:install odl-mdsal-broker
opendaylight-user@root>feature:install odl-flow-model
opendaylight-user@root>feature:install odl-flow-services
opendaylight-user@root>feature:install odl-openflowjava-protocol
opendaylight-user@root>feature:install odl-config-persister
opendaylight-user@root>feature:install odl-config-startup
opendaylight-user@root>feature:install odl-protocol-framework
opendaylight-user@root>feature:install odl-yangtools-models
opendaylight-user@root>feature:install odl-yangtools-data-binding
opendaylight-user@root>feature:install odl-yangtools-binding
opendaylight-user@root>feature:install odl-yangtools-common
opendaylight-user@root>feature:install odl-yangtools-binding-generator
opendaylight-user@root>feature:install odl-netconf-api
opendaylight-user@root>feature:install odl-netconf-mapping-api
opendaylight-user@root>feature:install odl-netconf-util
opendaylight-user@root>feature:install odl-netconf-impl
opendaylight-user@root>feature:install odl-config-netconf-connector
opendaylight-user@root>feature:install odl-netconf-netty-util
opendaylight-user@root>feature:install odl-netconf-monitoring
opendaylight-user@root>feature:install odl-openflowplugin-southbound
opendaylight-user@root>feature:install odl-openflowplugin-flow-services
opendaylight-user@root>feature:install odl-openflowplugin-drop-test
opendaylight-user@root>dropallpacketsrpc on
```

It is ready to the test.

A.3 CREDENCE

Download JBoss from the JBoss downloads page.

```
http://www.jboss.org/products/jbossas/downloads.
```

After you have downloaded the version you want to install, use the JDK jar tool (or any other ZIP extraction tool) to extract the jboss-4.0.4.zip archive contents into a location of your choice.
Before installing and running the server, you need to check your system to make sure you have a working Java 1.5 installation
Run the installer

```
$ java -jar jboss-4.0.4-installer.jar
```

Configure with your basic information
#Copy the "LibFluidStack-1.0.0-SNAPSHOT.jar" in the ../server/default/lib
#Copy the folder of "TopologyManager-sbb" and "LearningSwitch-sbb" into ../server/default/deploy
#Run JBoss

```
$ sh ../bin/run.sh
```

if have an error restart JBoss, maybe have problems with the firts run