Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR DEGREE THESIS

THESIS TITLE: Deploying and Testing of Central Office Re-architected as a Datacenter (CORD)

DEGREE: Bachelor Degree on Network Engineering

AUTHOR: Xavier Rins Lozano

ADVISORS: Anna Agusti Torra
David Rincon Rivera

DATE: June 15 th 2018

Título: Despliegue y Testeo de Central Office Re-architected as a Datacenter (CORD)

Autor: Xavier Rins Lozano

Directores: Anna Agustí Torra
David Rincon Rivera

Fecha: 15 de junio de 2018

Resumen

Durante los últimos años, el tráfico de datos en Internet se ha incrementado espectacularmente (se reportan incrementos de 100% en 8 años), lo cual provoca que la arquitectura de las redes de acceso basadas en xDSL y fibra se encuentren con el reto de escalar y adaptarse a las nuevas necesidades de los usuarios. Además, las centrales de las redes de acceso tienen una arquitectura poco coherente y unificada, formada por dispositivos complejos, caros y típicamente propietarios. Esto provoca que los operadores de red se estén enfrentando actualmente a diferentes problemas, tanto económicos como operacionales.

Una solución a estos problemas es Central Office Rearchitected as a Datacenter (CORD), que se encuentra en pleno desarrollo y que mediante la combinación de las tecnologías SDN, NFV y cloud computing pretende convertirse en la nueva arquitectura de las centrales de acceso.

Este proyecto se centra en explicar CORD al detalle, comparándolo con una arquitectura tradicional, instalando su última versión y creando documentación única para futuras aplicaciones. También se propone estudiar la posibilidad de modificar el algoritmo de asignación de recursos de Openstack, una de las piezas clave de CORD, con el objetivo de mejorar la eficiencia energética del sistema.

Como resultado, se ha instalado correctamente Cord in a Box 5.0 tanto en una máquina virtual proporcionada por Cloudlab como en un servidor físico de la Universitat Politècnica de Catalunya. A partir de esa instalación, se ha detallado y documentado CORD y Cord in a Box, generando una documentación que no estaba disponible hasta este momento.

También se ha instalado correctamente Cord in a Box 5.0 con un parche para añadir servidores extra al sistema. A partir de ahí, se ha creado nueva documentación, también inexistente hasta ahora. Además, se ha estudiado la posibilidad de modificar el algoritmo de asignación de recursos de Openstack y se ha iniciado la implementación de los cambios necesarios en el sistema para hacerlo posible.

Title: Deploying and Testing of Central Office Re-architected as a Datacenter (CORD)

Author: Xavier Rins Lozano

Advisors: Anna Agustí Torra
David Rincon Rivera

Date: June 15 th 2018

Overview

During the last years, data traffic on the Internet has increased dramatically (100% increases are reported in the last 8 years). Therefore, the architecture of access networks based on xDSL and fiber have been meeting the challenge of scaling up and adapting to the new needs of users. In addition, central offices do not follow a coherent or unified architecture, and are composed of of complex, expensive and typically proprietary devices. As a result, network operators are currently facing different problems, both economic and operational.

A solution to these problems is CORD, an architecture currently in development and research that includes SDN, NFV and Cloud technologies, and aims to become the new central offices architecture.

This project focuses on describing Residential CORD in detail, comparing it with a traditional architecture, installing its latest version and creating unique documentation for future applications or purposes. We also propose the application of an algorithm to improve the energy efficiency of Openstack (one of the key pieces of CORD).

Regarding the results obtained, Cord in a Box has been successfully installed in a virtual machine provided by Cloudlab. It has also been installed on a physical server at Universitat Politècnica de Catalunya. From that installation, CORD and Cord in a Box have been described in depth, creating documentation and explanations not publicly until now.

We have also installed Cord in a Box 5.0, adding a patch used to add extra servers to the system. From there, new documentation has been created, also non-existent until now. In addition, we have studied the possibily of modifying the Openstack resource allocation algorithm and we have initiated the implementation of the necessary changes in the system.

CONTENTS

INTRODUCTION

Long time ago, when a telephone call was to be made, rather than hearing a tone, what was heard was a real person (called an operator) asking you where and whom you wanted to call. That person manually tried to connect your line to the destination one. Nowadays, this work is done automatically, but the location of the devices is at the same place: at a Central Office (CO).

A Central Office is the physical place (usually a building) where all the phone and Internet communications of the subscribers of a telecommunication provider pass through.

In the last years, Central Offices have suffered a very high increase in data traffic [1]. In 1984, data traffic consumption around the world was 15 GB/month. In 2000, it was 75,250,200 GB/month. In 2014: 42,423,169,029 GB/ Month. This evolution can be seen in Fig. 1.

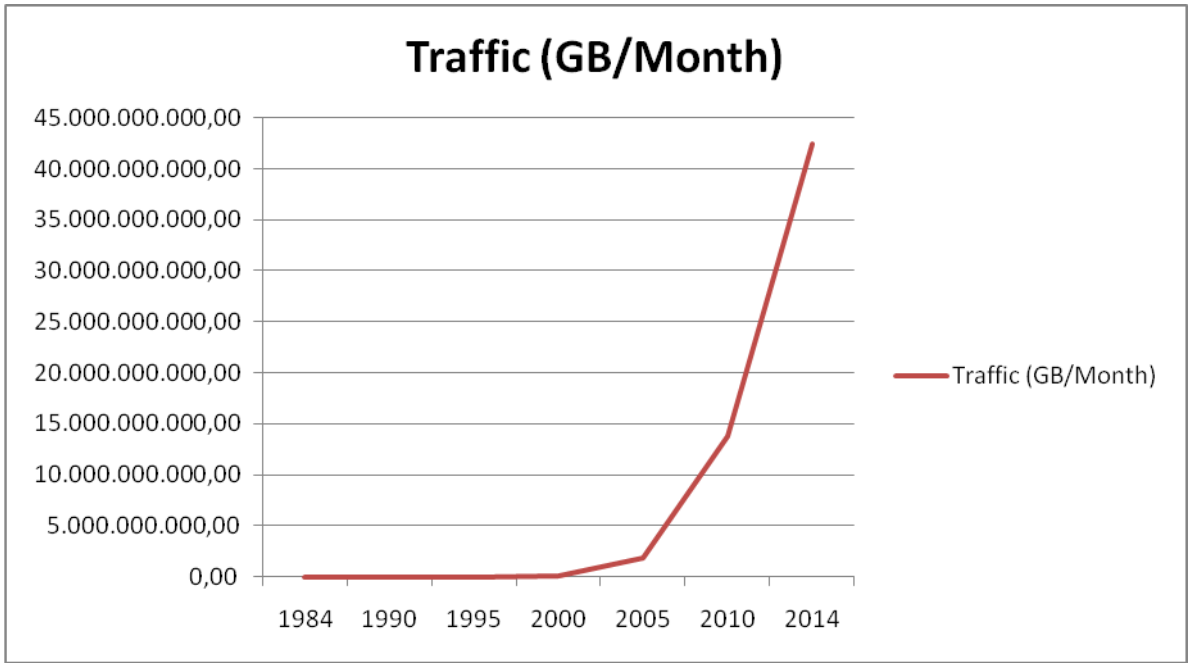


Fig. 1 Data traffic evolution since 1984 to 2014, from [1]

Currently, Central Offices contain expensive and complex devices, often coming from different manufacturers and without having a unified standard architecture. The rapid growth of Internet has caused network operators to have poor operational efficiency, high maintenance costs, and litte oportunity to offer user-centered services.

In a world where more and more bandwidth and improved services are needed, and where great scalability, flexibility and adaption are required, these factors

become a problem, and network providers are noticing it. But very rarely there is a problem without a solution, and this case is not an exception.

Central Office Re-architected as a Datacenter (CORD) is, as its name suggests, a new proposal for the central offices. Through the combination of differents technologies like NFV, SDN and cloud computing (explained in Chapter 1), CORD provides a unified and coherent architecture, solving the aforementioned problems.

CORD has application in the Residential, Enterprise, and Mobile markets. This variety of solutions, provided in a common hardware and software infrastructure, makes CORD a future proof solution.

Currently, there are seven projects under CORD [2]. Each project has its own community, roadmap and software components, but they all are coordinated for each release of the CORD reference implementation. These projects are:

- Residential CORD (R-CORD)
- Enterprise CORD (E-CORD)
- Mobile CORD (M-CORD)
- Analytics for CORD (A-CORD)
- Trellis: CORD network infrastructure
- XOS and the CORD controller
- VOLTHA: Virtual OLT Hardware Abstraction

All the community-related information can be found in [3]. All the documentation on the software in the reference implementation can be found in [4]

This project focuses on describing R-CORD in depth, from scratch, creating and providing new documentation and schemes in order to make it more understandable. We have also installed and described Cord in a Box, a simplified version of R-CORD, which runs on a single physical server or in a virtual machine. Moreover, we propose the integration of a new algorithm in order to improve the energy efficiency of Openstack in CORD, but the implementation has not been successful.

The structure of the project is as follows:

- Chapter 1: SDN, NFV & Cloud. The main technologies used by CORD are briefly explained here.

- Chapter 2: Introduction to CORD. In this chapter, the legacy Central Office architecture is compared with the architecture proposed by CORD.

- Chapter 3: Virtual POD (CORD in a Box). This chapter describes Cord in a Box and its installation, understanding R-CORD in detail.

- Chapter 4: CiaB with three compute nodes. In this chapter, a patch is added to CiaB in order to have three compute nodes in the system, similar to the reference implementation.

- Chapter 5: Improving Openstack – CORD efficiency. This chapter describes the aforementioned algorithm (the algorithm used to improve the energy efficiency of Openstack)

The document ends with the Conclusions and Future lines.

CHAPTER 1. SDN, NFV & CLOUD

1.1. SDN Basics.

Software Defined Network (SDN) is a network paradigm that separates the network's control and data planes and makes the control plane programmable. This simplifies network infrastructure and allows the development of economical white-box[1] switches that can be built using merchant silicon [5]. SDN is composed by three differentiated layers:

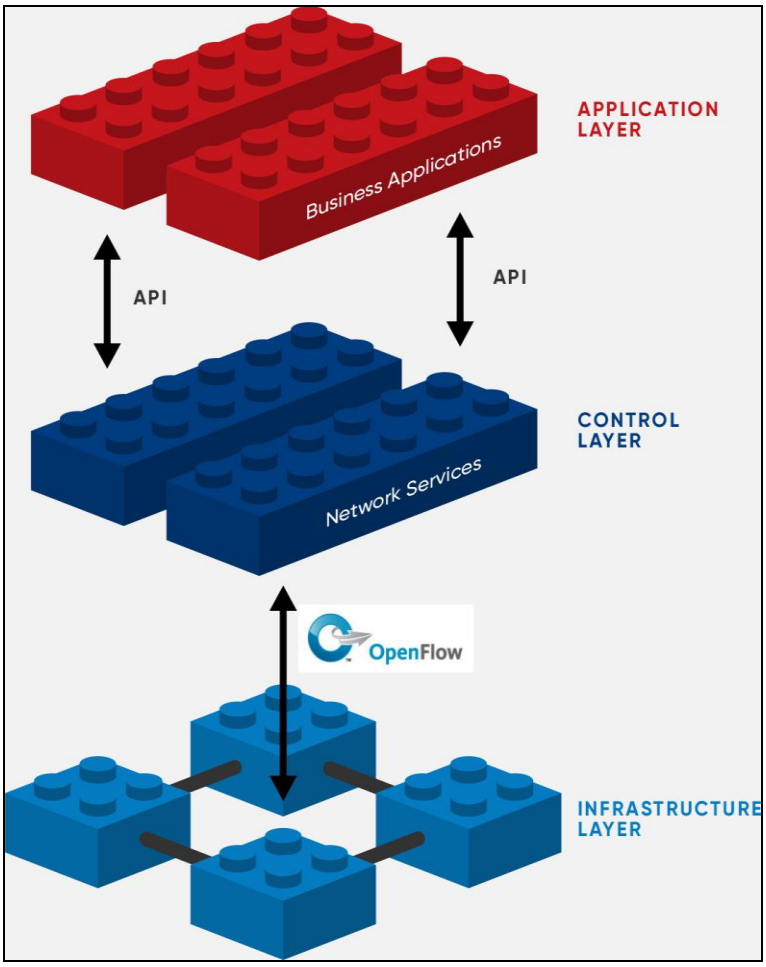


Fig. 1.1 SDN architecture, from [6]

- Infrastructure layer: It contains the hardware devices, which can be low-cost, white-box and no proprietary devices. They only need to support the OpenFlow[2] protocol.

[1] White box is a term used to describe the commoditization of IT devices by assembling off-the-shelf components.
[2] OpenFlow protocol is a communications protocol that gives access to the forwarding plane of a network switch of router. http://flowgrammable.org/sdn/openflow/

- Control layer: It contains the controllers, which keep the network intelligence. These controllers communicate with the hardware through the OpenFlow protocol. Also, these controllers provide an API (Application Programming Layer) in order to be programmed from the application layer.

- Application layer: It is the highest level layer. The behavior of the network is controlled from this layer. To achieve it, the API of the controller is used (this API can also be used by orchestration services).

1.1.1. ONOS

Open Network Operating System (ONOS) [7] is an open-source platform for service provider networks. The technology behind ONOS is backed by ON.Lab[3], as well as several leading service providers and vendors. It is a SDN operating system that delivers a highly available and scalable SDN control plane[4] featuring northbound and southbound open APIs that can be enabled by using several management, control, and service applications across mission-critical networks.

1.2. NFV Basics.

Network Function Virtualization (NFV) [8] is the process of moving network-related services, like load balancers or firewalls, from dedicated hardware into a virtualized environment. In summary, it moves the data plane from hardware devices to virtual machines running on commodity[5] servers. This replaces high-cost devices (like routers, firewalls, etc.) with commodity hardware and permits more agile software-based orchestration.

1.3. Cloud computing Basics.

Cloud computing is the practice of using a network of remote servers hosted on the Internet to store, manage and process data, rather than a local server or a personal computer. It brings different service models:

- Infrastructure as a service (IaaS): It refers to online services that provide high-level APIs used to dereference various low-levels of underlying network like phyisical computing resources, location, scaling, security, backup, etc. Openstack is an example of IaaS.

[3] ON.Lab: https://opencord.org/tag/on-lab/
[4] The control plane is the part of a network that carries signaling traffic and is responsible for routing, among other aspects.
[5] Commodity hardware, in an IT context, is a device or device component that is relatively inexpensive, widely available and more or less interchangeable with other hardware of its type.

- Platform as a service (PaaS): It provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. VMWare[6] is an example.

- Software as a service (SaaS): It reffers to online applications, as (for example) any web based service, like Gmail, Google Docs or Dropbox. In this case, all development, maintenance, updates, backups, etc, are responsibility of the provider.

1.3.1. OpenStack

OpenStack [9] is a free and open-source software platform for cloud computing, mostly deployed as an IaaS. It consists of interrelated components that control diverse pools of processing, storage, and networking resources throughout a data center. Users either manage it through a web-based dashboard, through command-line tools, or through RESTful web services.

The main components are:

- Nova (computing): This module provides computing services. It is used for hosting and managing cloud computing systems.

- Neutron (controller): It provides networking services in Openstack between interface devices managed by other Openstack services.

- Keystone (authentication): It is the identity service used by Openstack for authentication and high-level authorization. Different openstack services like Cinder, Nova, Glance, etc. will be authenticated by Keystone.

- Glance: It is used to provide image storage and metadata service.

- Cinder: It refers to the block storage service for OpenStack. Cinder virtualizes the block storage devices and presents the storage resources to end users.

- Ceilometer: It is a component of the Telemetry project. Its data can be used to provide customer billing, resource tracking, and alarm capabilities across all OpenStack core components.

Fig. 1.1 provides more details on how the different components are communicated.

[6] VMware provides cloud computing and platform virtualization software and services. Its webpage is https://www.vmware.com/es.html.

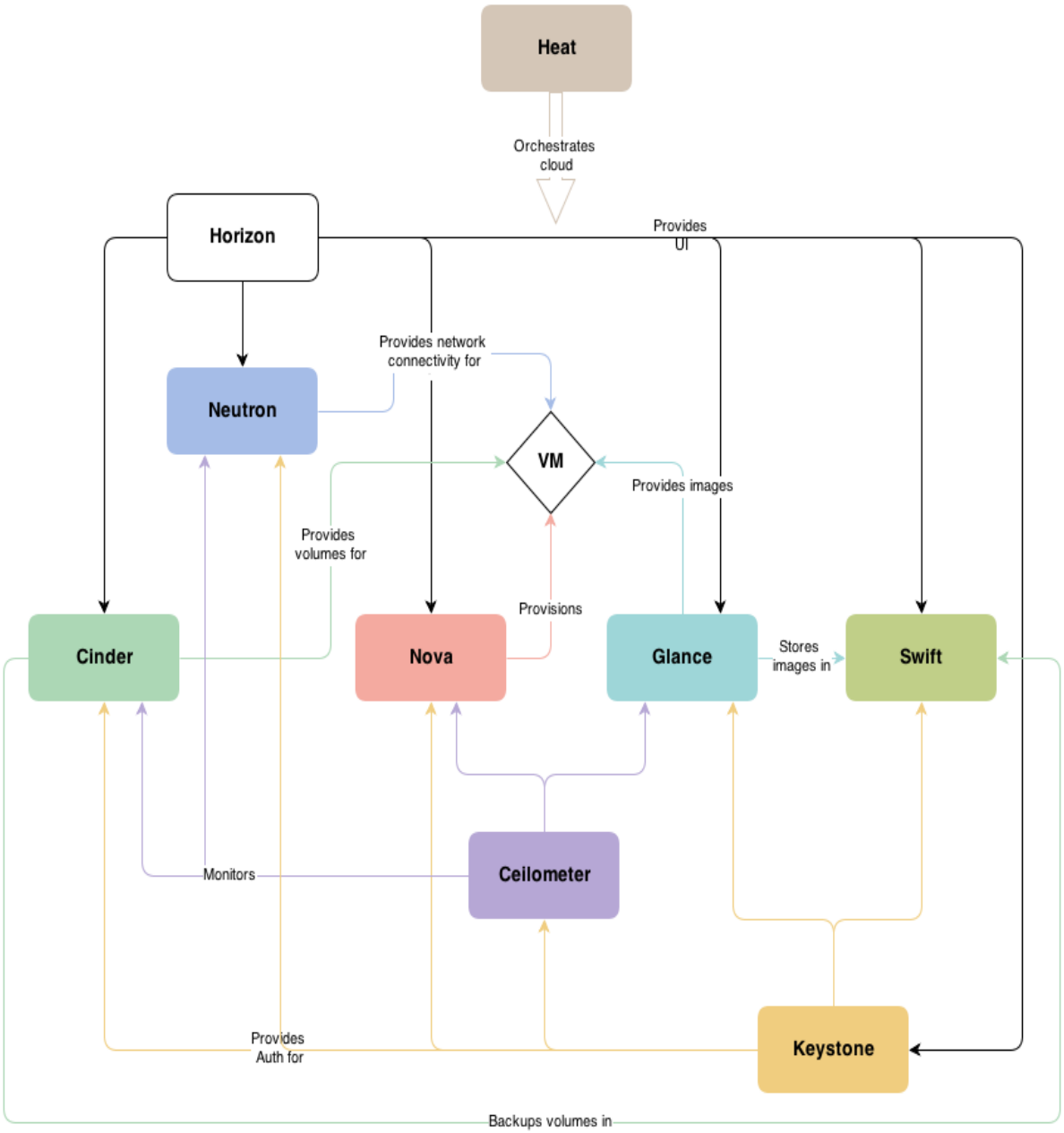


Fig. 2.1 Openstack components diagram, from [10]

CHAPTER 2. INTRODUCTION TO CORD

2.1. Legacy architecture for Central offices

2.1.1. Architecture

Until now, the central office of a telecommunications operator (such as, for example, Telefónica[7]) has had an architecture similar to the one shown in Fig. 2.1 and Fig. 2.2 (for the case of a FTTH access network; for the case of Fiber access network, there is a local loop).

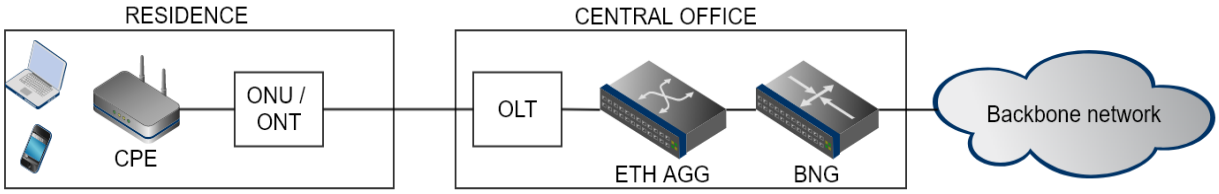


Fig. 2.1 High level scheme of the central office architecture

Fig. 2.1 shows the five main elements of a typical access network.

- **Customer Premises Equipment (CPE)**. Router at home
- **Optical Network Unit / Termination (ONU/ONT).** Converts optical signals transmitted via fiber to electrical signals which are then sent to individual subscribers
- **Optical Line Terminal (OLT)**. This is the first element that is located in the central office, and it has two main functions. The first one is to perform conversion between the electrical signals used by the service provider's equipment and the fiber optic signals used by the passive optical network. The second one is to coordinate the multiplexing between the ONU/ONT devices.
- **Ethernet Aggregation Switch (EthAgg). I**t is responsible for aggregating traffic from the different OLTs.
- **Broadband Network Gateway (BNG).** It provides access to the Backbone Network.

Fig. 2.2 illustrates the architecture with more details, such as the introduction of DSL devices.

[7] Telefónica is a Spanish multinational broadband and telecommunications provider with operations in Europe, Asia, and North, Cenral and South America. It is also part of the CORD initiative.

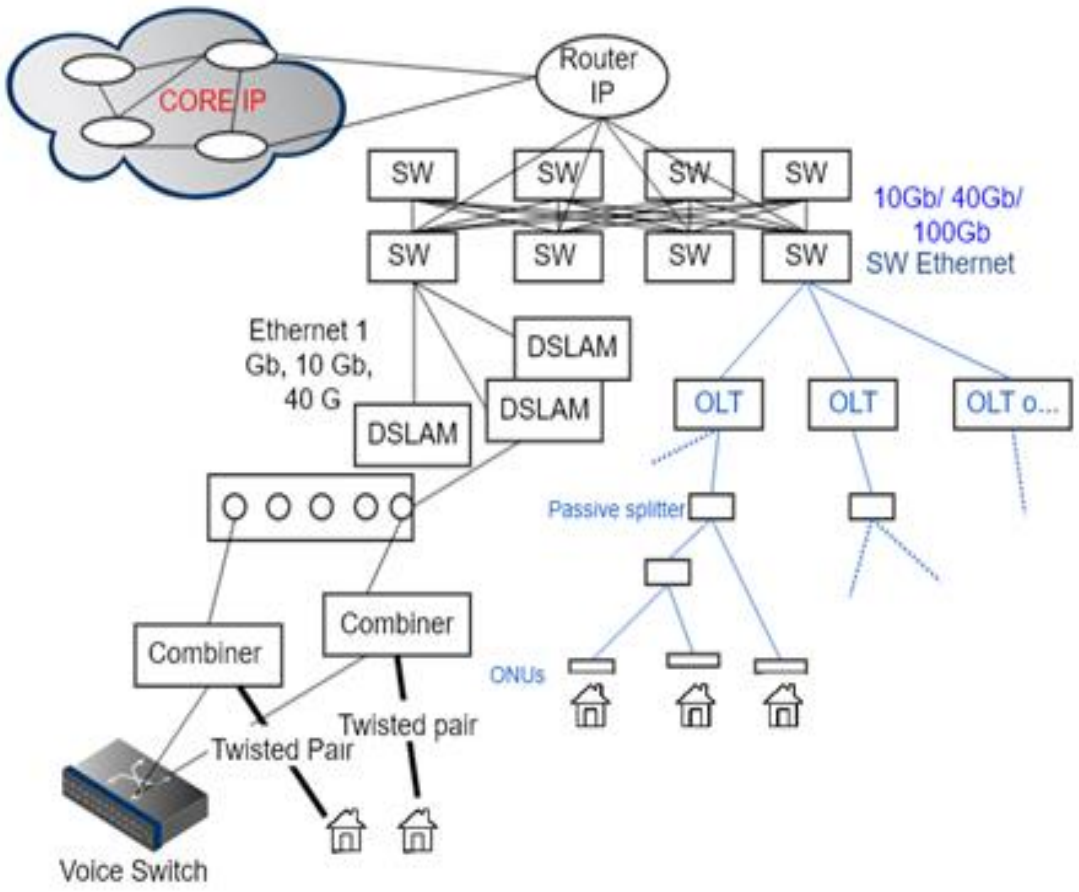


Fig. 2.2 Current Central Office architecture

As shown in Fig. 2.2, the Ethernet Aggregation Switch communicates with other Ethernet Agreggation Switch (normally following a fat tree topology[8]) in the central office. Another thing that can be observed is that every OLT typically supports 32/64 clients. More elements that can be seen are:

- TDM-based voice switches.
- DSLAMs batteries that are connected to each of the copper twisted pairs.

2.1.2. Disadvantages

The previously described architecture uses closed and proprietary[9] hardware. Moreover, network operators are facing increasing challenges in meeting bandwidth demands and providing enhanced services. Therefore, network

[8] Fat tree network is a universal network for provably efficient communications. More information can be found here:
https://www.cs.cornell.edu/courses/cs5413/2014fa/lectures/08-fattree.pdf

[9] Proprietary hardware is a computer hardware whose design and interface are controlled by the vendor, often under patent or trade-secret protection, and typically is not interoperable with products for other vendors, since they do not follow any standard.

operators are limited from both the economies of scale (infrastructure) and agility (the ability to rapidly deploy and elastically scale services).

Currently, telecommunication central offices contain specific-built devices, assembled over many years, with a not very coherent or unifying architecture. This creates them a problem because it is a source of significant capital (CAPEX) and operational (OPEX) expenditures, as well as it limit their innovation capabilities.

In 2017 AT&T saw data traffic increase by 100% in the last eight years [11]. At the same time, introducing a new feature often takes months (waiting for the next vendor product release) and sometime years (waiting for the standardization process to run its course).

Regarding the CPE we can say that it is currently distributed to thousands of homes assigned to the same Central Office, which makes it an operational burden. This is a big problem when updating a service requires updating some hardware, for example.

Regarding the BNG, BNG routers are very expensive and complex. These routers have aggregated a lot of functionalities provided by the Central Office, making them very difficult elements to be incorporated in an agile and effective system.

2.2. Towards the future: CORD

Central Office Re-architected as a Datacenter (CORD) is a new proposal for the Central Offices backed by AT&T, ONF and other operators and vendors. Part of this section has been obtained and summarized from [11].

2.2.1. From the past to the future

Because of the disadvantages mentioned in Section 2.1, network operators are looking for different ways to increase their scalability and their agility. To achieve this, two steps are needed:

1- Disaggregate and virtualize elements, transforming every complex and proprietary hardware-based element to software running in commodity hardware.

2- Provide a framework in which the resultant disaggregated elements can be added, producing a coherent end-to-end system and managing these disaggregated elements as a whole. This is also known as “service orchestration”.

These two principles define CORD.

2.2.2. Central Office Re-architected as a Datacenter (CORD)

CORD [12] is an architecture for the network central office that combines software-defined networking (SDN), network functions virtualization (NFV), and elastic cloud services to build cost-effective, agile access networks. The mission of CORD is to bring datacenter economies and cloud agility to service providers for their residential, enterprise, and mobile customers. The reference implementation of CORD will be built from commodity servers, white-box switches, disaggregated access technologies (e.g., vOLT[10]), and open source software (e.g., OpenStack, ONOS, XOS).

2.2.3. CORD = SDN + NFV + cloud

Unifying these three technologies, CORD provides something better than what they can give separately.

CORD can simplify the networking infrastructure, but it can also provide new services that can be offered to customers. It supports conventional SaaS, but also extends the cloud to include fiber to the home and wireless access as elastic access as a service. It supports legacy network functions in Virtual machines (VMs), but also disaggregates functionality into finer-grained elements.

In summary, the objective of CORD is not only to replace hardware devices with their more agile software-based counterparts, but also make the Central Office an integral part of every telecommunication's cloud strategy, enabling a big collection of services, including access for residential, mobile and enterprise customers.

2.2.4. Hardware of the reference implementation

As aforementioned, CORD runs in commodity and rackable hardware. This rackable unity is called pod.

CORD defines a reference implementation for pods [13], but this is one of the many hardware configurations that can be done. CORD also accepts any access technology (e.g. 10GPON, G.FAST, etc.).

The reference implementation of a CORD pod consists of 3 different type of elements:

- 6 Servers: Open Compute Project (OCP)-qualified QUANTA STRATOSS210-X12RS-IU servers, each configured with 128 GB of RAM, 2 × 300 GB HDDs, and a 40GE dual-port NIC.

[10] Virtual OLT (vOLT) replaces proprietary OLT devices with a combination of commodity hardware and open source software.

- 6 Switches: OCP-qualified and OpenFlow-enabled Accton 6712 switches, each configured with 32 × 40GE ports, and doubling as both leaf and spine switches in the CORD fabric.

- 2 I/O Blades: OCP-contributed AT&T Open GPON — NFV OLT Line Card. Celestica-manufactured Optical Line Termination (OLT) “pizza boxes” that include merchant silicon OLT MAC chips from Microsemi. This is a 1u-blade that includes 48 × 2.5 Gb/s gigabit passive optical network (GPON) interfaces and 6 × 40GE uplinks

The configuration scheme of these elements is shown in Fig. 2.3

2.2.5. Software of the reference implementation

In terms of software, the reference implementation uses 4 open source projects, as shown in Fig. 2.4. These four open source projects are:

OpenStack [9]: is responsible of the cluster management that provides capacity of IaaS. It is also responsible for creating and provisioning the VM and the virtual networks (VNs).

Docker [14]:provides a container-based media to implement and interconnect services.

ONOS [7]: is the operating system that manages both the switching software and the fabric physical switching. It also runs a set of applications which implement subscriber services and switching fabric management. ONOS interconnects VMs (implementing VNs and managing flows through the fabric switching) and provides a platform to host control programs that implement CORD services (such as vOLT or vRouter).

XOS [15]: is the framework used to assemble and compose services. XOS unifies the service infrastructure (provided by OpenStack), the service plane (provided by ONOS) and any data plane or cloud services (running in VM or containers). In short, is responsible of the step 2 explained in section 2.2.1.

To make CORD useful, it has to be as compatible as possible. It is for this reason that this reference implementation supports services running in:

- Virtual machines.
- Containers running directly in Bare Metal[11].
- Containers inside the VM.

[11] Bare Metal environment is a computer system or network in which a virtual machine or container is installed directly on hardware rather than within the host operating system. The term Bare Metal refers to a hard disk, the usual medium on which a compuer’s OS is installed.

2.2.6. Schemes of the reference implementation

Fig. 2.3 describes the reference implementation [12].

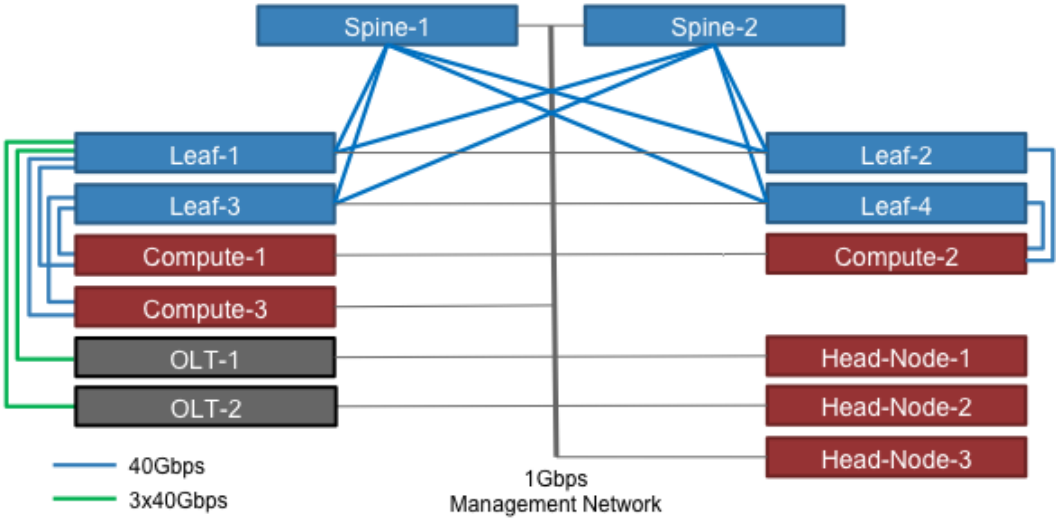


Fig. 2.3 Hardware pod with servers, switches and I/O blades, from [13]

Servers, switches and OLT blades are assembled in two virtual racks (single physical rack) as shown in Fig. 2.3. The servers and the OLT blades are interconnected by the "leaf-spine" switching fabric (similar to a fat-tree topology), which consists of two spine switches and two leaf-switches per virtual rack. Also, as shown in Fig. 2.3, leaf-2 is connected to an upstream router, and the OLT-blades are connected via GPON to the ONTs and home routers.

The servers run Ubuntu LTS 14.04 and include Open vSwitch (OVS)[12]. The switches run the open source Atrium software stack [3], which includes Open Network Linux, the Indigo OpenFlow Agent (OF 1.3), and Broadcom's OpenFlow Data Plane Abstraction (OF-DPA), layered on top of Broadcom merchant silicon.

As it can be seen in Fig. 2.4, XOS assembles multi-tenant services, ONOS hosts control applications, and OpenStack/Docker manage compute instances.

The system includes 3 types of services: Cloud service (for example: CDN), data plane service (for example: vSG) and control plane service (for example: vOLT and vRouter). The BNG element has been dissagregated: it is subsumed as a combination of vOLT, vSG, vRouter and the underlying switching fabric. These elements are described below.

[12] Open vSwitch [16] is an open-source implementation of a distributed virtual multilayer switch. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks.

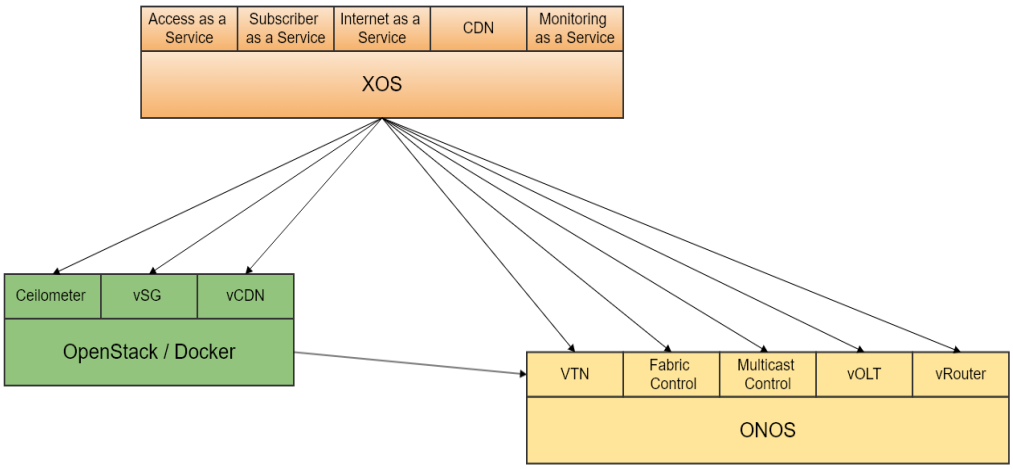


Fig. 2.4 OpenSource software components in CORD

- vOLT (Virtual Optical Line Termination): control program running in ONOS. Implements Access as a Service (AaaS). Every tenant acquires a subscriber VLAN. It also implements a subscriber-based authentication and other functions of the legacy OLT.

- vSG (Virtual Subscriber Gateway): data plane function scaled through a set of containers. It implements Subscriber as a Service (SaaS), where every tenant acquires a subscriber group.

- vRouter (Virtual Router): control program running in ONOS. It implements “Internet as a Service” (IaaS), where every tenant acquires a routing subnet.

- CDN (Content delivery network): it is a cloud scalable service deployed by networks operators.

CORD transforms the legacy CPE, the OLT and the BNG elements shown in Fig. 2.4 to the vCPE, vOLT and vBNG elements, respectively, shown in Fig. 2.5.

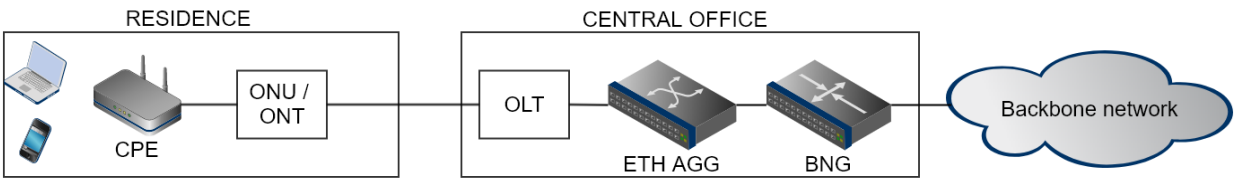


Fig. 2.4 Legacy elements

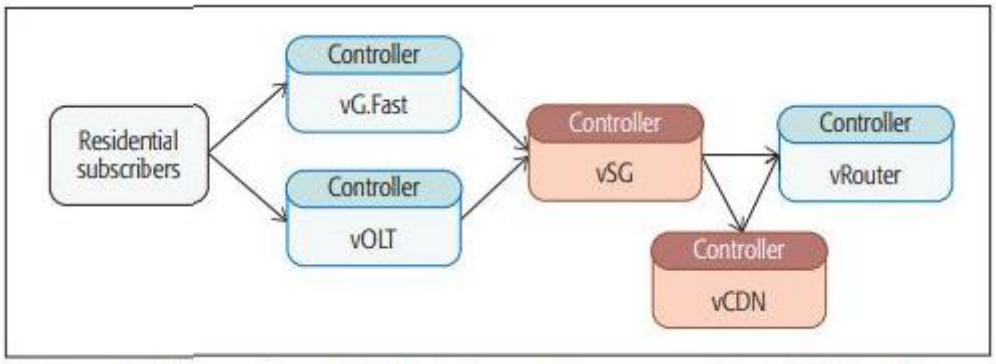


Fig. 2.5 virtualized elements – transforming to CORD, from [11]

In conclusion, the two steps commented in section 2.2.1 results in the legacy Central Office being re-architected into the service scheme shown in Fig. 2.5

The end-to-end packet flow will be studied in Chapter 3.

CHAPTER 3. VIRTUAL POD: CORD IN A BOX (CiaB)

3.1 Definition

Some problems faced by people who are interested in CORD is that it is very volatile and dynamic (there have been 3 new versions in a year) . In addition, in the official CORD pages, there is very few documentation or diagrams explaining R-CORD, its functions, the packet routing, the rules that ONOS sends to the OVS, the function of the different existing services, etc.

The goal of this chapter is to provide a clearer vision of R-CORD, through the installation of Cord in a Box, analyzing it, and creating new and unique documentation, explaining in depth different parameters not explained until now.

As a brief definition, we can say that CiaB is a simplified virtual CORD pod, running in multiple virtual machines on a single physical server.

3.1.1 Differences between CORD and CiaB

Obviously, this virtual pod has limitations compared to a full physical pod with multiple servers. The main difference between CORD and CiaB is that CiaB only virtualizes the head node and the compute nodes. It does include neither the Trellis fabric (leaf-spine switches[13]) nor the access devices.

CiaB is very useful to understand how packets flow from the subscriber (missing the vOLT) to the vSG, going through the Open vSwitch, and how the system is able to identify subscriber traffic[14] and deliver the packets to the corresponding vSG[15]. It is also useful to see how OpenStack creates and eliminates virtual machines, and what criteria follows to do it.

3.1.2 Requirements, experiments, installation and GUIs:

3.1.2.1 Requirements

As shown in CiaB Open Cord Guide [17], there are some requirements to run the installation:

- 64-bit AMD64/x86-64 server, with:
    - 48 GB+ RAM

[13] Leaf-spine is a two-layer network topology composed of leaf switche (to which leaf servers and storage connect) and spine switches (to which leaf switches connect). Leaf switches mesh into the spine, forming the access layer that delivers network connection points of servers.
[14] Subscriber traffic refers to the traffic created by a CORD user in his home.
[15] Virtual Subscriber Gateway (vSG) refers to the virtualized version of CPE. It runs a bundle of subscriber-selected functions. But does so on commodity hardware located in the Central Office rather than on the customer's premises.

    - 12+ CPU cores
    - 200GB+ disk
- Access to the internet (no enterprise proxies)
- Ubuntu 14.04.5 LTS
- User account with password-less sudo capabilty (explained below).

3.1.2.2 Attempts

We have tried to complete the installation in three different environments:

1- Virtual machine deployed in an Openstack private cloud environment (UPC-EETAC cloud): *This experiment was not successful because of the openstack environment. When the Openstack environment was installed, the system broke and errors appeared. We tried to increment the timeout by modifying an Ansible script, but we did not succeed.*

2-Virtual machine deployed in CloudLab: *successfully installed*

3-Physical server deployed at UPC: *successfully installed*

Annex A includes an installation guide of the basic Cord in a Box. Annex B describes the different user interfaces which can be accessed.

3.1.3 High level explanation

CiaB is composed of multiple Vagrant[16] VMs inside a dedicated physical node or a VM:

- Corddev: it is used to manage the installation.

- Head1: The head node runs the Openstack, ONOS and XOS services. Openstack runs in LXD Containers. Onos and XOS run in Docker containers. The client container (emulating subscriber's home) will also run on this node.

- Compute1: Compute1 is controlled by OpenStack. It will run the different virtual machines instantiated by OpenStack (nova) and XOS. This node will also run an OVS controlled by ONOS to route the traffic of each client to its corresponding vSG and vice versa.

[16] Vagrant [18] is an open-source software product for building and maintaining portable virtual software development environments (VirtualBox, Docker containers, VMware...)

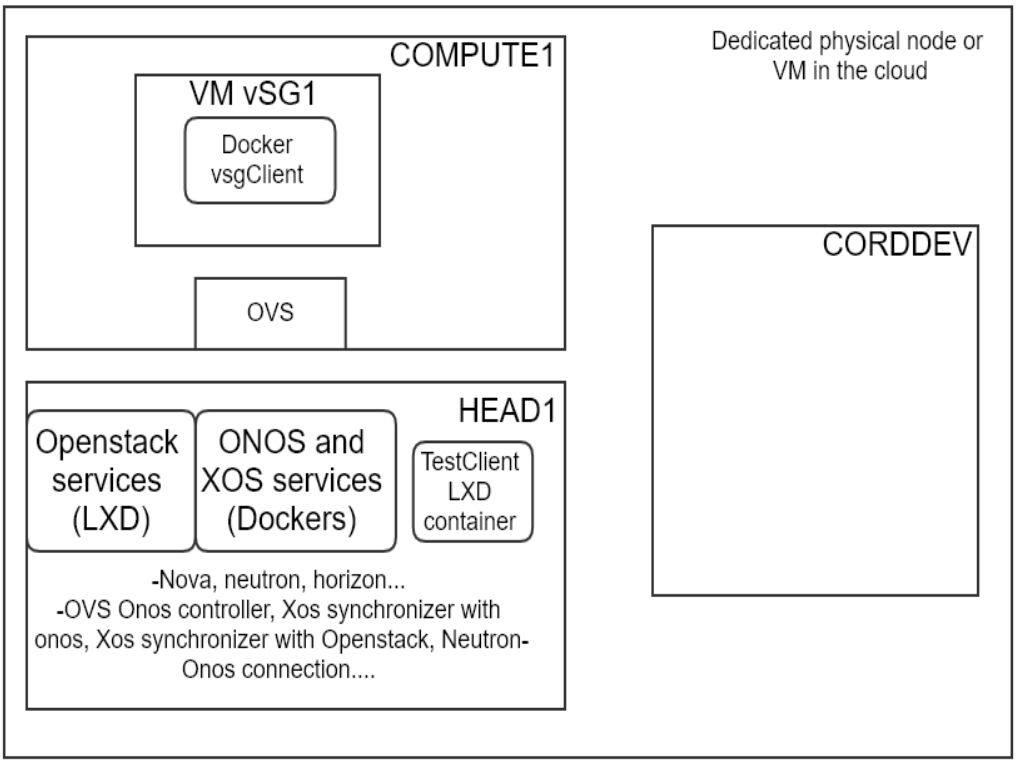


Fig. 3.1 High level scheme of CiaB

We will now describe what happens when a subscriber sends traffic to the global Internet. Also, the different packet flows involved in the communication will be analyzed.

3.1.4 From subscriber to the global Internet. How traffic is routed from the test-client container.

To explain this, a ping has been sent to Google from the test client container while capturing the traffic in the different interfaces. It is important to remember that "Test Client LXC container" is simulating a residential user. Fig. 3.4 shows how CiaB routes the traffic of a subscriber client to the global Internet.

The 4 different types of packets shown in the following figures have been captured in the "fabric" interface executing `tcpdump –i fabric –w "filename"`[17]. Then, the file has been imported to Wireshark[18] and analyzed.

[17] tcpdump is a common packet analyzer that runs under the command line. It allows the user to display the packets veing transmitted or received.
[18] Wireshark is a network packet analyzer. It tries to capture newtork packets and display their data as detailed as possible

-Packet type 1:

```
Frame 11: 106 bytes on wire (848 bits), 106 bytes captured (848 bits)
Ethernet II, Src: Xensourc_b2:d7:a5 (00:16:3e:b2:d7:a5), Dst: c6:7d:80:14:65:6d (c6:7d:80:14:65:6d)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 222
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 111
Internet Protocol Version 4, Src: 192.168.0.225, Dst: 172.217.12.3
Internet Control Message Protocol
```

Fig. 3.2 Packet type 1: TestClient to vSG docker

-IP source*:* 192.168.0.225 (testClient)
-IP destination: Google
-MAC source: finished with d7:a5 (belonging test Client)
-MAC destination: finished with 65:6d (belonging vsG docker)
-VLan IDs: 222,111

-Packet type 2:

```
Frame 12: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: 02:42:0a:07:01:03 (02:42:0a:07:01:03), Dst: OpenNetw_06:01:01 (a4:23:05:06:01:01)
Internet Protocol Version 4, Src: 10.7.1.3, Dst: 172.217.12.3
Internet Control Message Protocol
```

Fig. 3.3 Packet type 2: vSG to head1

IP source: 10.7.1.3 (vSG docker)
IP destination: Google
MAC source: finished with 01:03 (belonging eth0 vSG docker client)
MAC destination: finished with 01:01 (belonging head node interface 10.7.1.1)

-Packet type 3:

Inverse of packet 2

-Packet type 4:

Inverse of packet 1

The explanation will be described step by step:

1-Test Client to OVS
2-OVS to vSG VM
3-vSG VM to vSG docker
4-vSG docker to the Head node
5-Head node to the global Internet
6-Global Internet to the Head node
7-Head node to vSG docker
8-vSG docker to vSG VM
9-vSG VM to OVS
10-OVS to test Client

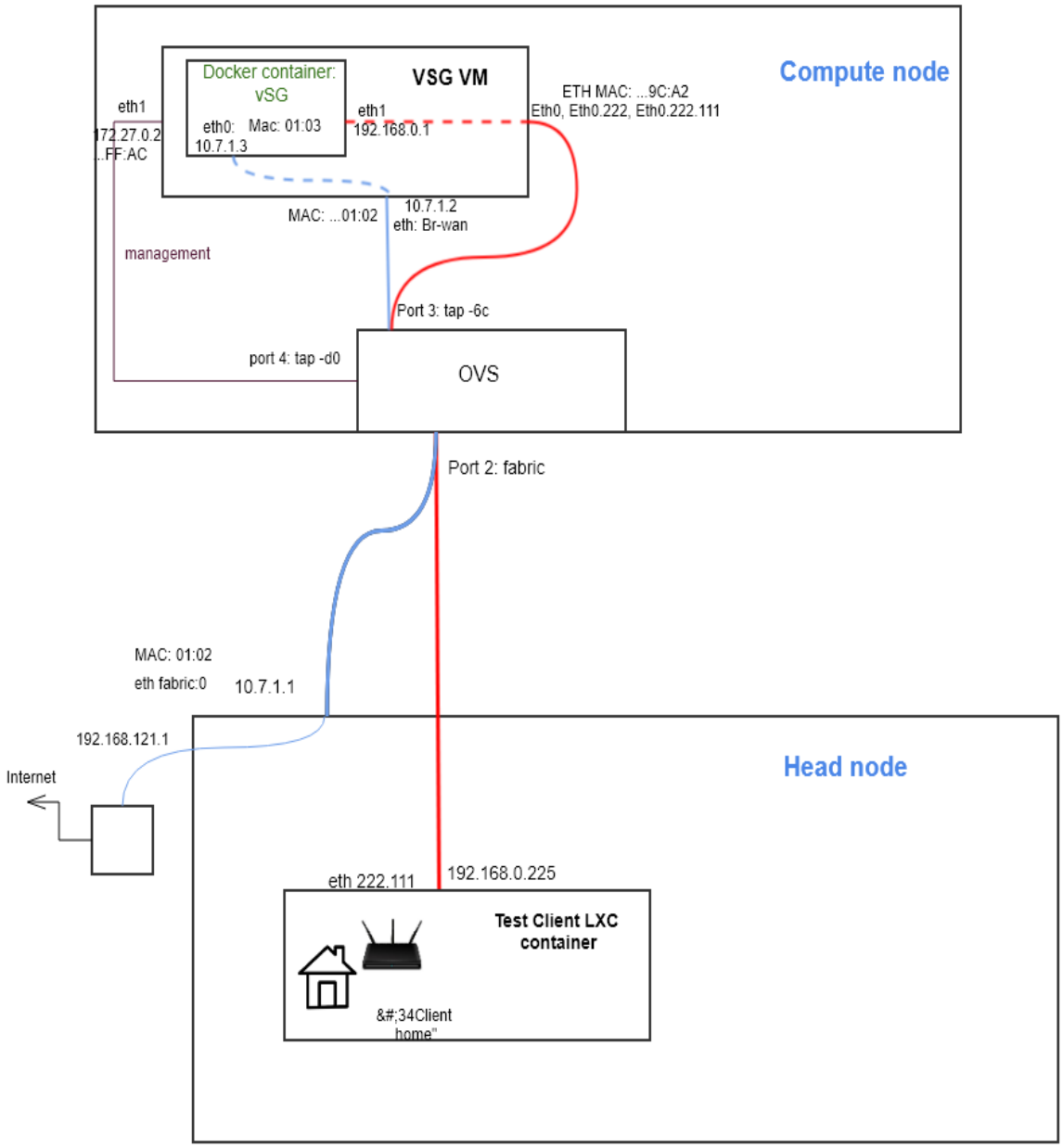


Fig. 3.4 Scheme of CiaB routing the traffic

1- TestClient to OVS:

When the client wants to send traffic to the Internet, the packet leaves the test Client container via eth222.111 and goes to the port 2 (fabric interface) of the OVS. At this step, the packet type 1 (aforementioned) appears. This packet has two VLan tags. The tag with VLAN ID 222 is emulating the vOLT ID, and the tag with VLAN ID 111 is emulating the subscriber home inside the PON.

In the next steps it is explained what rules has the OVS to route the traffic properly. Fig.3.5 shows what happens in CORD and what CiaB tries to emulate:

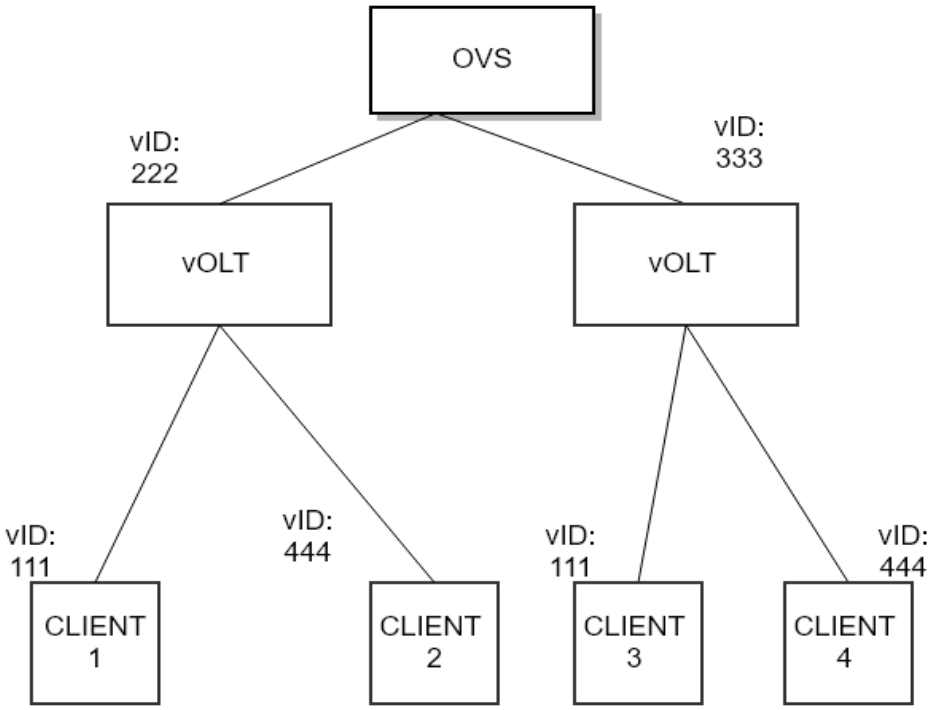


Fig. 3.5 Scheme of what CiaB tries to emulate.

So, as it can be seen, the Test Client Container is simulating client 1 of vOLT 1 (VLAN tag 111.222), and it is being shown the packet after the vOLT.

2- OVS to vSG VM:

Now, the packet arrives to the OVS. If the OVS is observed through the ONOS controller, it can be found the next rule applying this step:

```
Criteria: IN_PORT: 2, VLAN_VID: 222
Action: OUTPUT: 3
```

Due to this rule, the OVS sends the packet through Port 3 (tap -6c). In the scheme, this route is represented by the red line. The packet arrives at the vSG VM and is received by the interface eth0.222.111.

3- vSG VM to vSG docker:

This is an easy step. The interface eth0.222.111 of vSG VM is directly connected with the eth1 of the docker inside the vSG. The IP of the interface eth1 is 192.168.0.1 (VLAN tags are removed by the interface eth0.222.111).

4- vSG docker to head node:

At this point, the vSG docker removes the VLAN tag and acts like a NAT inside the system. Redirects the packet through eth0 and changes the IP to 10.7.1.3 (eth0 IP).

To make it easier, it is explained in a way that the vSG docker is connected with the eth Br-wan of the OVS (blue line in the scheme).

But actually in CiaB (more complicated and strange) the vSG docker sends the packet to the OVS with the IP destination 10.7.1.2. The OVS, with the next rule,

```
Criteria: ETH_TYPE: ipv4, IPV4_DST: 10.7.1.2/32
Action: VLAN_PUSH: vlan, VLAN_ID: 500, ETH_DST: ...9C:A2, OUTPUT: 3
```

adds a VLAN tag 500 (not useful in the present) and sends the packet to 10.7.1.2 through interface -..9C:A2[19].

Obviously, the packet goes again to the OVS, who has the next rule:

```
Criteria: VLAN_VID: 500
Action: VLAN_POP:500, output: 2
```

The OVS sends the packet to the Head node via Port 2 (fabric). If the packet is captured at this moment, it is shown the packet type 2 (explained before).

5- Head node to the Internet:

The packet arrives to interface eth0: fabric of the head node, with the IP 10.7.1.1. At this point, it is sent to 192.168.121.1 (in the global node, outside of head node) and finally, the packet goes to the Internet. This is done this way because CiaB does not include the trellis switching fabric.

Steps 6,7,8,9 and 10 are exactly the inverse of 1,2,3,4 and 5 (respectively).

3.2 Adding subscribers

In this section a subscriber will be added to CiaB in order to see what happens with the different elements of the system. It will be done from the Orchestrator[20] XOS. As previously mentioned (see section 3.1.2), XOS can be acceded by HTTP.

It is important to keep in mind that creating a new subscriber will not create another Test Client Container in Head1 (emulating a new user's home) but will create the corresponding vSG docker in the corresponding vSG VM with the corresponding rules in the OVS.

Two steps are needed to add a subscriber.
1: Creating an RCORD Profile with specific Service ID.
2: Creating a vOLT tenant with the previously Service ID and the corresponding S tag and C tag.

[19] "…9C:A2" means that the MAC address of that interface finish with this numbers: 9C:A2.
[20] Orchestration is the automated arrangement, coordination, and management of computer systems, middleware, and services.

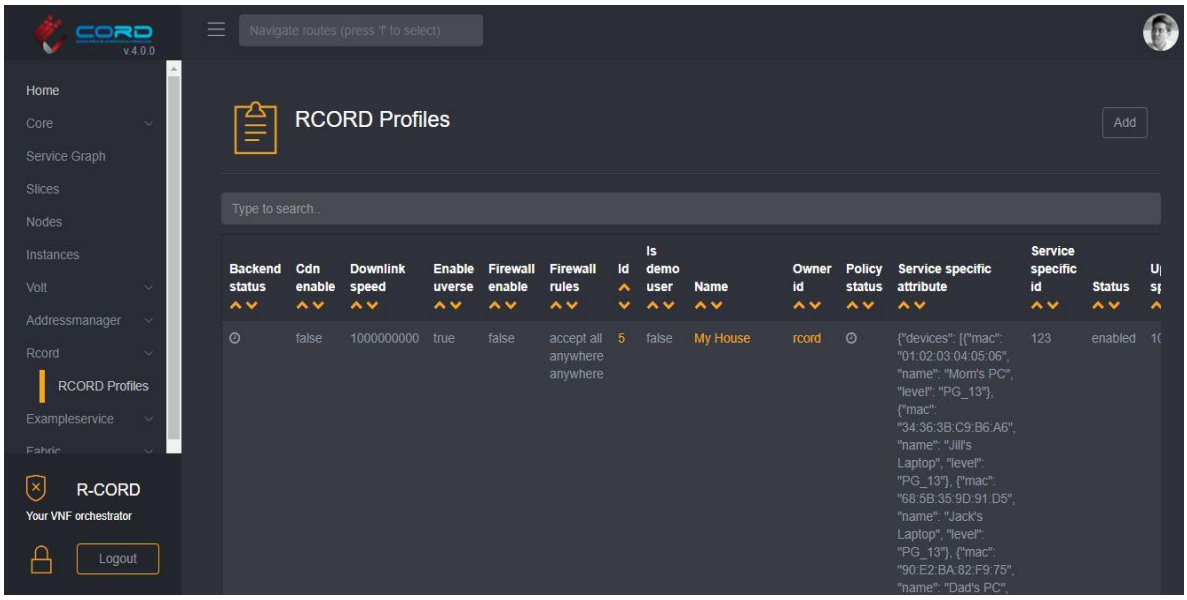


Fig. 3.6 XOS User interface.

Once authenticated to XOS, we have to click to Rcord - RCORD Profiles. In this case, “Xavi’s house”, with Service Id 12, was created. The next step is to create a vOLT tenant.

3.2.1 Adding subscriber in the same vOLT (same s_tag)

If we check vOLT - vOLT tenants, it can be seen the vOLT tenant with C tag 111, S tag 222 and Service ID 123. So, as previously mentioned, the subscriber “My House” has vOLT id 222 (S tag) and the tag inside the PON would be 111.

Now, a new vOLT tenant is going to be added in the same vOLT (222). To do this, it is necessary to click “add” and complete the form with the corresponding tags and Service ID:

When this vOLT tenant is added, the system creates a new vSG docker inside the same vSG VM. So, the system creates one vSG VM per vOLT and one vSG docker inside the vSG VM per client inside the same vOLT. As it has been seen before, the OVS only routes the traffic to the vSG VM (not to the docker inside the vSG VM). So the OVS only adds rules when new a vOLT (different S tag) is created.

In order to route the traffic inside the vSG VM to the different dockers, the system creates different interfaces in the vSG VM connecting to the different dockers (eth0.222.111, eth0.222.333, etc). This can be seen in Fig. 3.8.

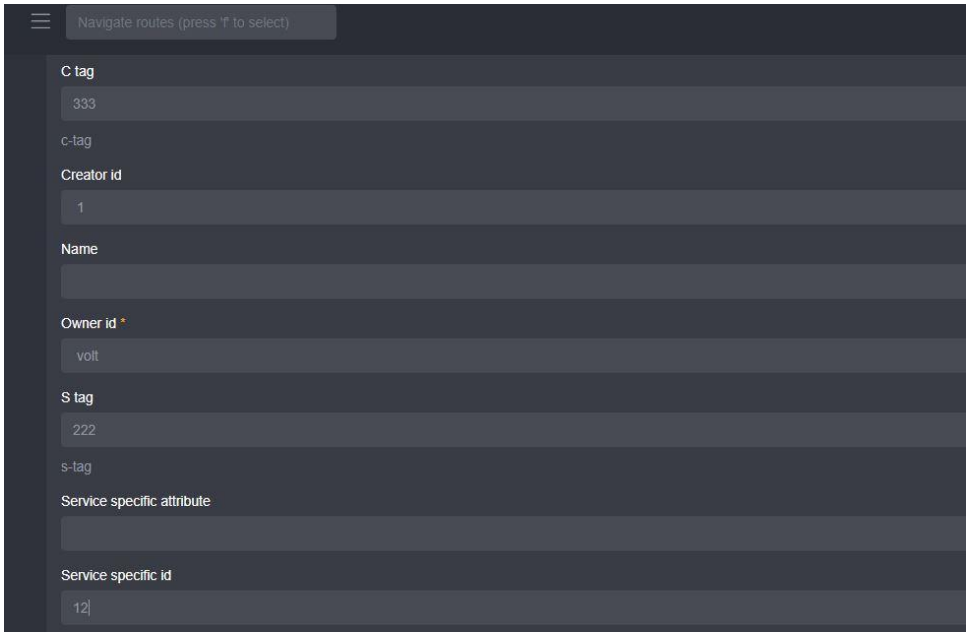


Fig. 3.7 Adding a new vOLT tenant in XOS.

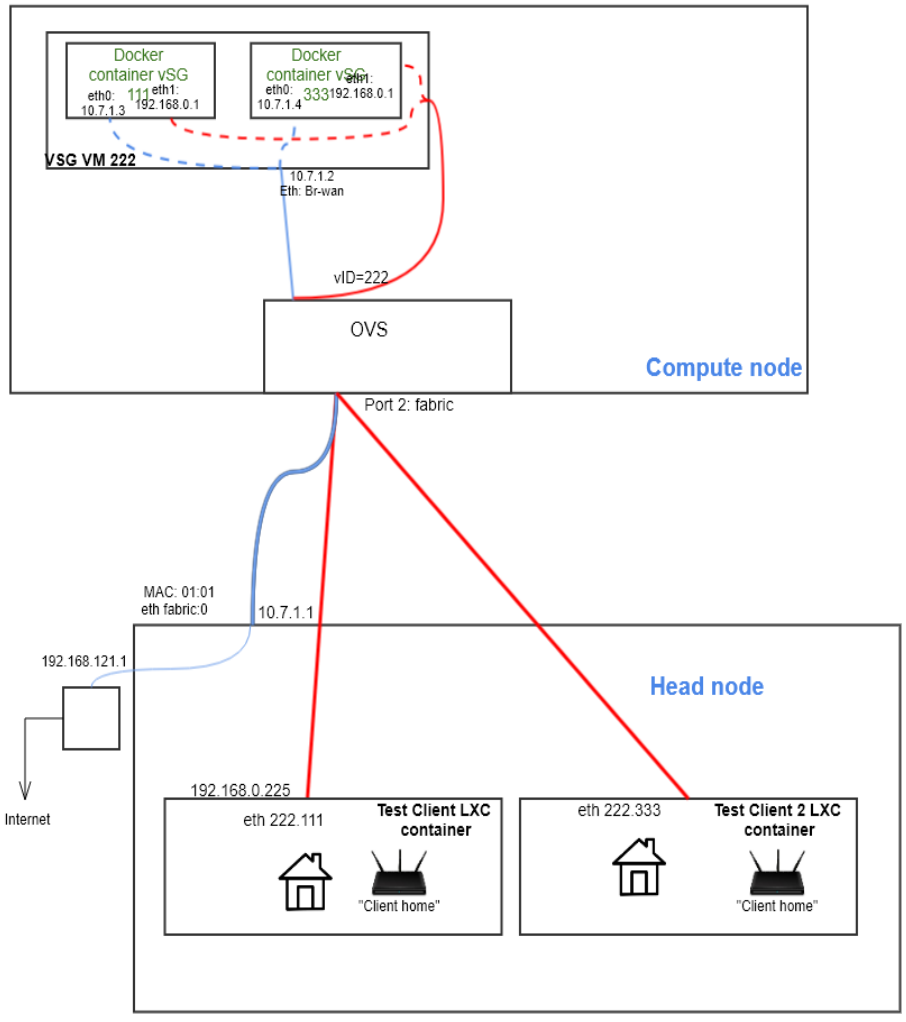


Fig. 3.8 Adding subscriber in the same vOLT

It is important to say that the new Test Client LXC container has to be created manually (it is not installed by default).

3.2.2 Adding subscriber in a different vOLT (different s_tag)

Now, a new subscriber is added in a different vOLT. To do it, the same steps are needed as before, but with a small difference: It is necessary to put a different S tag than before.

In our case, we have defined:

- S tag: 333
- C tag: 666

The final diagram is shown in Fig. 3.9.

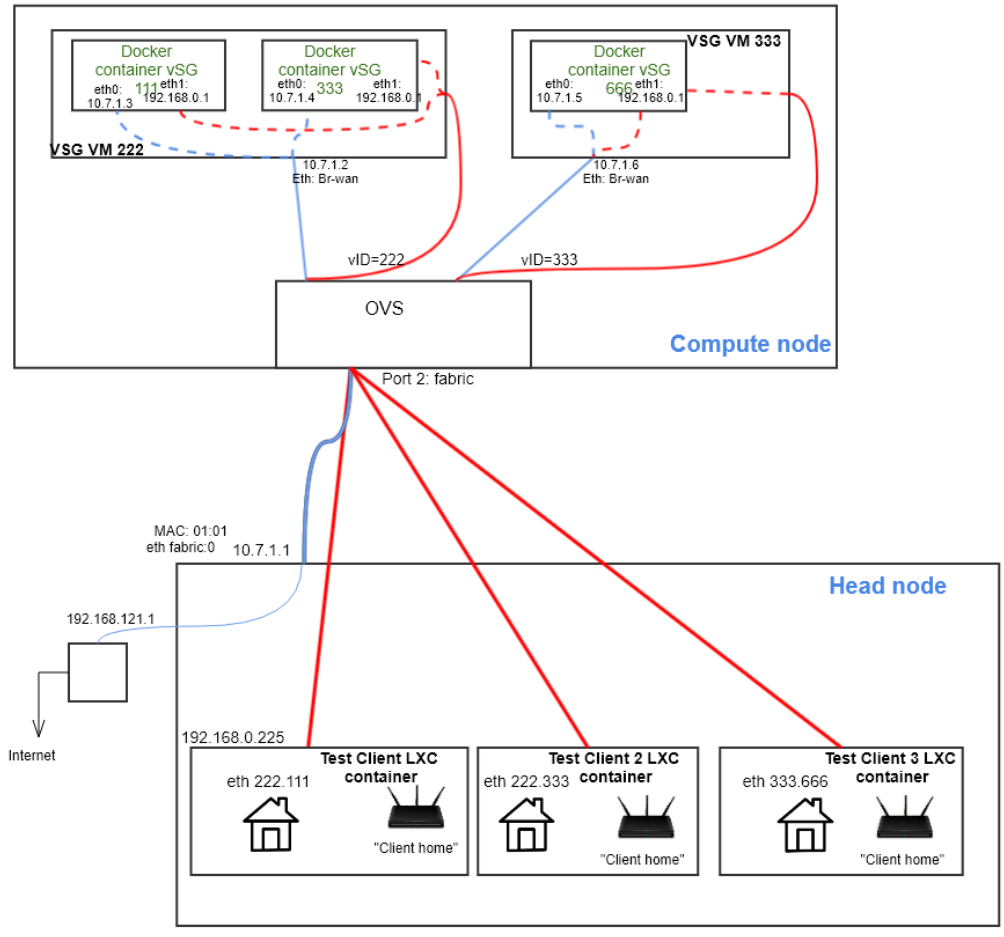


Fig. 3.9 Adding subscriber in a different vOLT

As commented before, the system now creates a new VM and a new docker inside this VM. The “Fine Grained Tenancy Graph” of XOS is shown in Fig. 3.10:

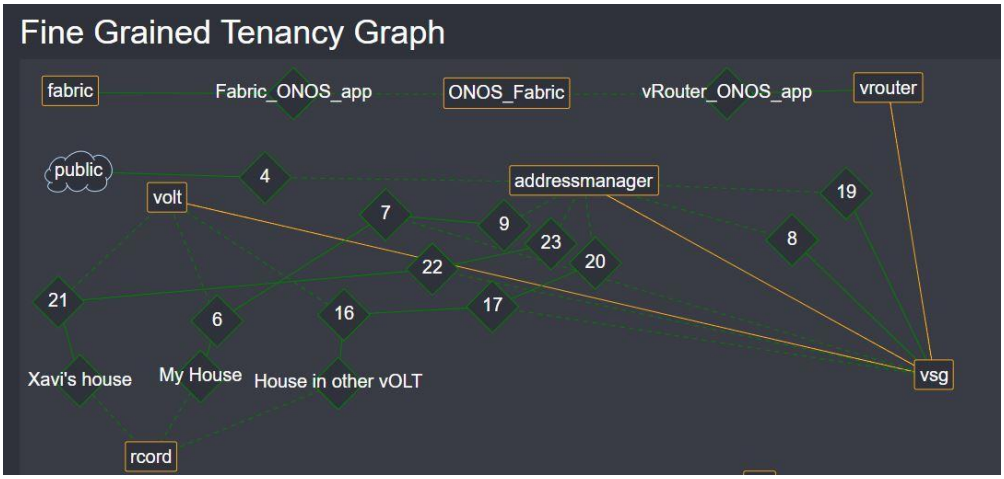


Fig. 3.10 XOS Fine Grained Tenancy Graph

As it can be observed, the system has three RCORD Profiles: “Xavi’s house” (S Tag 222), “My House” (S tag 222), “House in other vOLT” (S tag 333).

All of these RCORD Profiles belong to the vOLT service.

3.2.3 XOS-OpenStack-ONOS communication

It has been commented previously that OpenStack creates the VM or the Dockers when a new vOLT is added.

OpenStack knows when to create the VMs because when a new subscriber is added, XOS makes queries to the OpenStack API.

In more detail, when a subscriber is added manually, XOS executes functions through an own API REST. For example, if the subscriber is added in a new vOLT, in order to create a new virtual machine, XOS calls a code function that is in the LXContainer of Nova, inside the Head1 (passing as parameters different values and options of this virtual machine that is going to be created).

In conclusion, OpenStack is a slave of XOS. There is an LXContainer of ONOS-XOS. There is also a plug-in that communicates ONOS with Neutron.

3.2.4 ONOS-OVS communication (OpenFlow traffic)

We now describe how ONOS updates the rules of the OVS through Open Flow 1.3 protocol.

When the ONOS Docker receives the order, sends Open Flow Protocol (OFTP) messages through eth0, with IP destination 172.21.0.1 (interface of compute node), which resends the packets by the eth mgmbr (IP destination 10.1.0.14). On the other hand, when the packet comes from the OVS with IP source 10.1.0.14, the head node resends the packet through eth Br-9b (with IP

destination 172.21.0.2). Fig. 3.12 illustrates the ONOS-OVS communication in CiaB.

10.1.0.14: IP address of Compute Node.
10.1.0.1: IP address of Head Node.

The next packets have been captured in the eth br-...9b (subnet 172.21.0.0/24) when a new subscriber has been added in a new vOLT:

2435	35.728719	172.21.0.2	10.1.0.14	OpenFlow	154	Type: OFPT_FLOW_MOD
2436	35.728794	172.21.0.2	10.1.0.14	OpenFlow	138	Type: OFPT_FLOW_MOD
2437	35.728816	172.21.0.2	10.1.0.14	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
2438	35.728831	172.21.0.2	10.1.0.14	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
2439	35.729448	10.1.0.14	172.21.0.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
2440	35.729478	10.1.0.14	172.21.0.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY

Fig. 3.11 ONOS-OVS adding flows

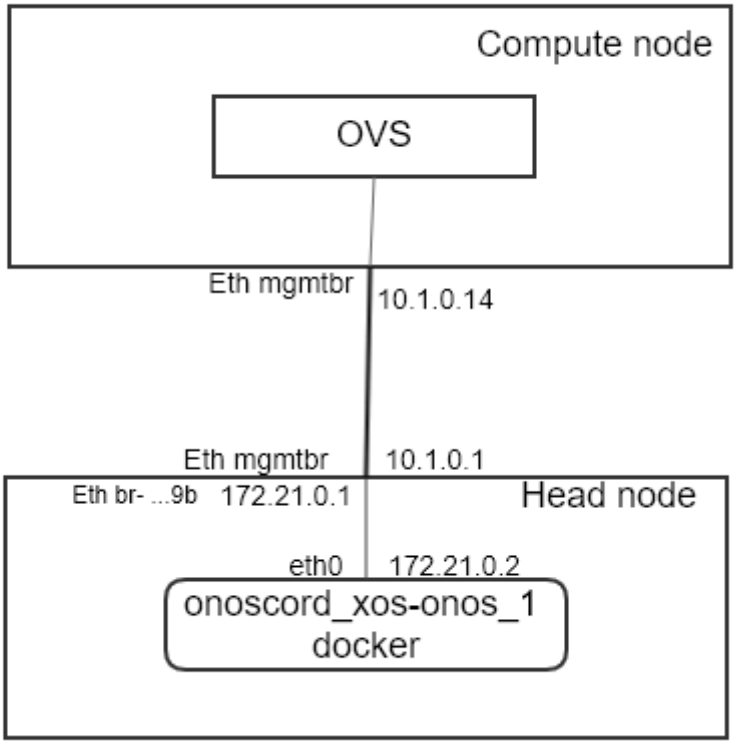


Fig. 3.12 ONOS-OVS communication diagram

We described here only the packets related with the new subscriber and its new VM vSG. It is described the exchange messages from the packet 2457 to 2462.

In packet 2457, an OFPT_FLOW_MOD message is sent from the controller to the OVS.

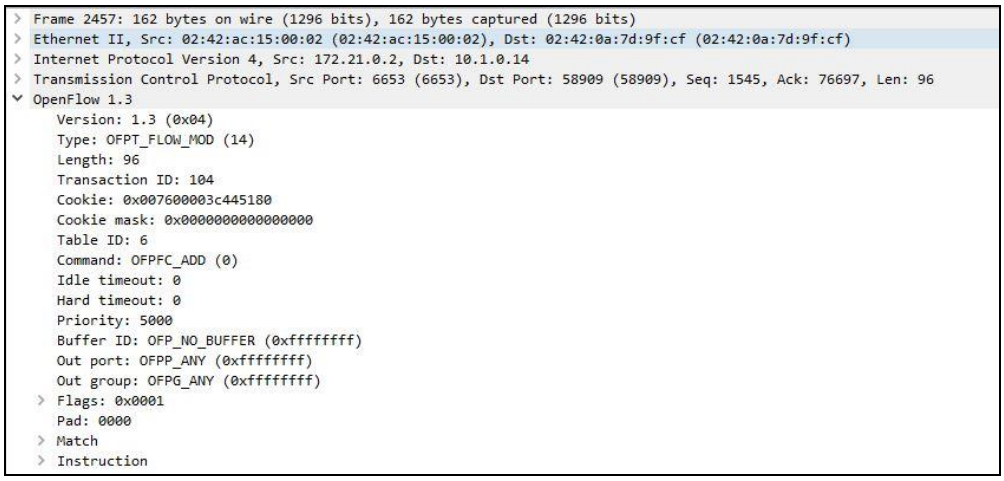


Fig. 3.13 ONOS-OVS Frame 2457

The command = “OFPFC_ADD” means that this flow has to be added (it is a new flow) and the idle and hard timeouts are 0 (permanent rule). Now, if the match and action fields are displayed, we can see this:

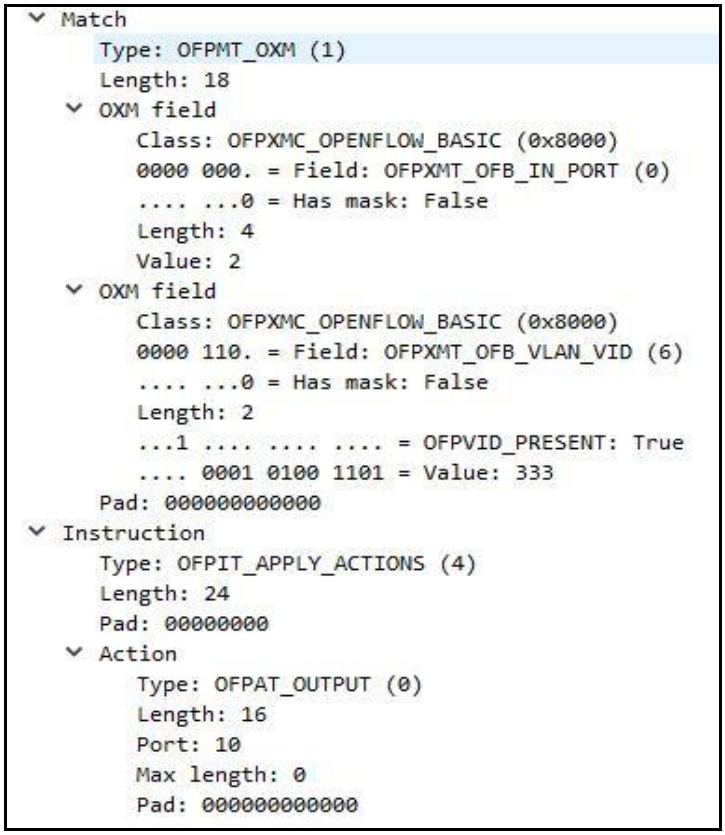


Fig. 3.14 ONOS-OVS Frame 2457, match and action fields

As it shown, there are two requirements (match OXM field) to do the action that is observed. These requirements are:

```
-IN_PORT= 2
-VLAN_VID= 333
```

```
Action:
-OUTPUT_PORT: 10
```

So, when a packet arrives at port 2 of the OVS (fabric interface) and it has VLAN vID 333 (new subscriber S tag = vOLT S tag), the OVS output the packet through port 10 (to its VM vSG). If the next packet is observed, the opposite is seen.

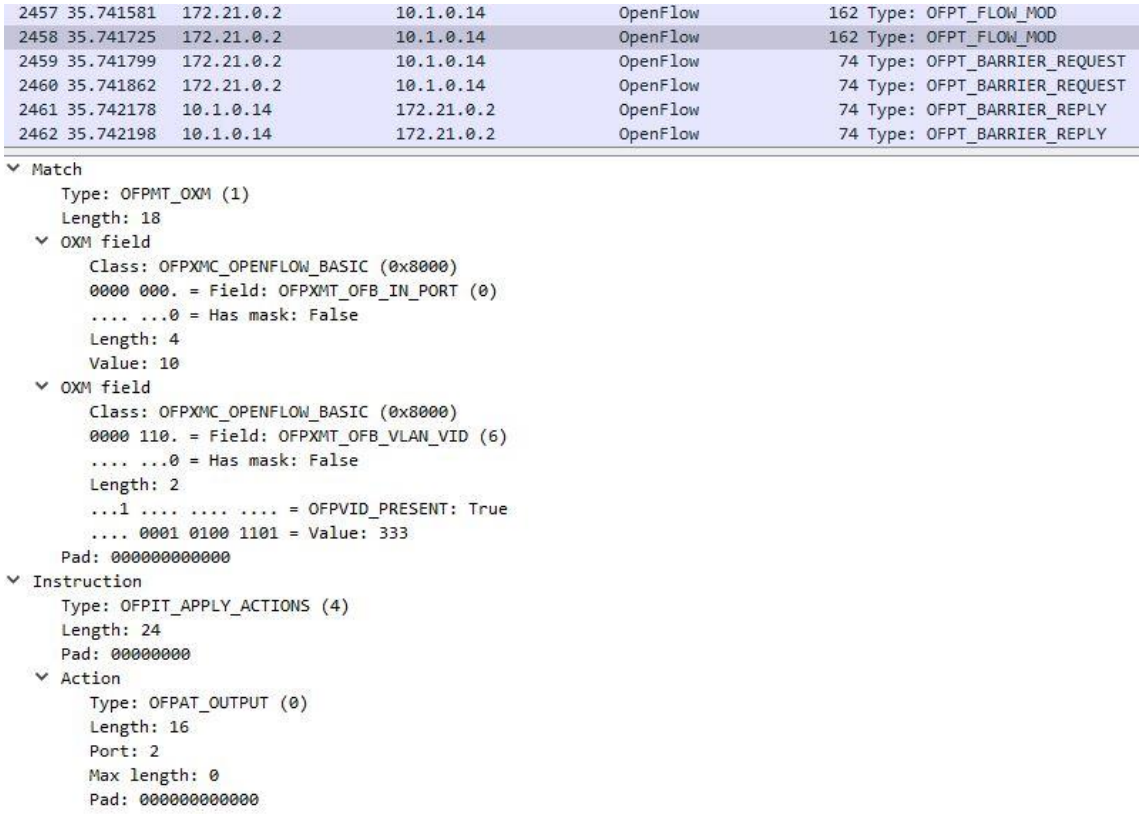


Fig. 3.15 ONOS-OVS Frame 2458, match and action fields

Now, as expected, the packet arrives at port 10 with the VLAN vID 333 (from the VM vSG) and it is routed through port 2 (to the client subscriber).

Then, the controller sends two packets “Barrier Request” to ensure that the previous two messages are completed. In order to confirm that, the OVS sends two “Barrier Reply” (also seen in Fig. 3.15).

3.3 Monitoring consumption

In this section, we have monitored the RAM and the disk usage memory before, during and after different stress process. Among other things, this has been

very useful for us because it has saved us a lot of time. Thanks to this, we have been able to know if we have had an error because of exhaustion of the available memory. It is also very useful for Chapter 5, where we will try to improve the Openstack energy efficieny

The tests have been done with “top” and “df” tools and it has been performed different times in order to have more reliable data. The test has been performed on a virtual machine provided by Cloudlab, allocated in Utah, and on a physical server at Universitat Politecnica de Catalunya. We have taken samples when there was 0 subscribers, one subscriber and two subscribers.

3.3.1 Cloud

3.3.1.1 Zero subscribers

When CiaB is installed, there is no client. In this step, the following data has been measured:

```
top - 07:51:45 up  1:29,  2 users,  load average: 3.03, 2.84, 2.97
Tasks: 257 total,   2 running, 255 sleeping,   0 stopped,   0 zombie
%Cpu(s): 15.0 us,  1.4 sy,  0.0 ni, 83.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  65714520 total, 65335236 used,   379284 free,   104856 buffers
KiB Swap:        0 total,        0 used,        0 free. 41234000 cached Mem
```

Fig. 3.16 System consumption

As it can be seen,“%Cpu(s)” row shows the CPU state percentages:

Us (user): CPU user time.
Sy (system): CPU kernel time.
Id (inactive, idle): No process requiring processor time.

In this case:

Us: 15%
Sy: 1.4%
Id: 83.5%

It can also be seen “Mem” and “Swap” rows. This portion consists of two lines which are expressed in Kibibytes (KiB).

“Mem” reflects physical memory, classified as total, used, free, buffers. This is the RAM of the machine, phyisical pieces of hardware that provide Random Access Memory.

“Swap” is the virtual memory which can be a file or a partition on the hard drive that is essentially used as extra RAM.

In this case (in Gigabyte):

-Total RAM: 67.29 GB.
-Used RAM: 66.9 GB
-Free RAM: 0.38 GB
-Buffers: 0,107
Swap memory is 0 in cloud CiaB.

Fig. 3.17. figure shows the disk usage:

```
xarilo@node:~/cord/build$ df -h
Filesystem                                      Size  Used Avail Use%
udev                                             32G  4.0K   32G   1%
tmpfs                                           6.3G  852K  6.3G   1%
/dev/nvme0n1p1                                   16G  5.1G  9.8G  35%
none                                            4.0K     0  4.0K   0%
none                                            5.0M     0  5.0M   0%
none                                             32G     0   32G   0%
none                                            100M     0  100M   0%
ops.utah.cloudlab.us:/proj/cord-testdrive-PG0   150G   23G  128G  15%
ops.utah.cloudlab.us:/share                      97G  8.7G   81G  10%
/dev/nvme0n1p4                                  213G   63G  140G  32%
```

Fig. 3.17 Disk memory

With Cloud CiaB we cannot extract conclusions of disk memory because there are many files and configuration that are shared with the other users. But if everything is added...100 GB are used. Actually, our machine is the “cord-testdrive”, and only is using 23GB, but this is not true because, as aforementioned, there are many files shared with the other users.

3.3.1.2 One subscriber

We now run the test-example that adds 1 client into the system, obtaining the following output (while executing the test):

```
top - 08:11:56 up  1:49,  3 users,  load average: 2.85, 2.31, 2.45
Tasks: 265 total,   2 running, 263 sleeping,   0 stopped,   0 zombie
%Cpu(s): 17.7 us,  2.2 sy,  0.0 ni, 80.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  65714520 total, 65336920 used,   377600 free,    73316 buffers
KiB Swap:        0 total,        0 used,        0 free. 37920004 cached Mem
```

Fig. 3.18 System consumption when executing test

-Us: 17,7%
-Sy: 2,2%
-Id: 80,1%
-Total RAM: 67.29 GB.
-Used RAM: 66.905 GB
-Free RAM: 0.386 GB
-Buffers: 0.075 GB

Once this client is installed, the “df” command shows exactly the same as before:

```
xarilo@node:~/cord/build$ df -h
Filesystem                                  Size  Used Avail Use%
udev                                         32G  4.0K   32G   1%
tmpfs                                       6.3G  848K  6.3G   1%
/dev/nvme0n1p1                               16G  5.1G  9.8G  35%
none                                        4.0K     0  4.0K   0%
none                                        5.0M     0  5.0M   0%
none                                         32G     0   32G   0%
none                                        100M     0  100M   0%
ops.utah.cloudlab.us:/proj/cord-testdrive-PG0  150G   23G  128G  15%
ops.utah.cloudlab.us:/share                  97G  8.7G   81G  10%
/dev/nvme0n1p4                              213G   69G  134G  34%
```

Fig. 3.19 Disk memory

3.3.1.3 Two subscribers

In the section 3.3.2, a new client has been added running an example-test. Now, a client is added by the Orchestrator, and we can see the output (it has been captured during the process):

```
top - 08:25:27 up  2:03,  2 users,  load average: 1.21, 1.50, 1.90
Tasks: 256 total,   2 running, 254 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.1 us,  0.9 sy,  0.0 ni, 88.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  65714520 total, 65282836 used,   431684 free,    77492 buffers
KiB Swap:        0 total,        0 used,        0 free. 37645784 cached Mem
```

Fig. 3.20 System consumption when there are two clients

-Us: 11,1%
-Sy: 0,9%
-Id: 88,1%
-Total RAM: 67.29 GB.
-Used RAM: 66.8 GB
-Free RAM: 0.442 GB
-Buffers: 0.07 GB

3.3.2 UPC CiaB

CiaB has also been installed on our UPC physical server and the same “ZeroClient” monitoring tests have been performed. The results are:

.

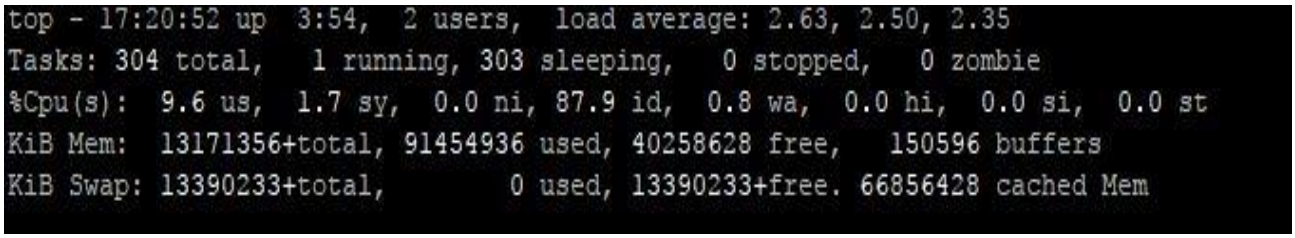


Fig. 3.21 System consumption in UPC CORD

-Us: 9,6%
-Sy: 1,7%
-Id: 87,9%
-wa: 0.8% (waiting for an I/O operation to complete)
-Total RAM: 134,487 GB.
-Used RAM: 93.64 GB
-Free RAM: 41.22 GB
-Buffers: 0.15 GB

Swap memory:

-Total: 13.7 GB
-Used: 0 GB
-Free: 13.7 GB

“Df” tool (Disk memory):

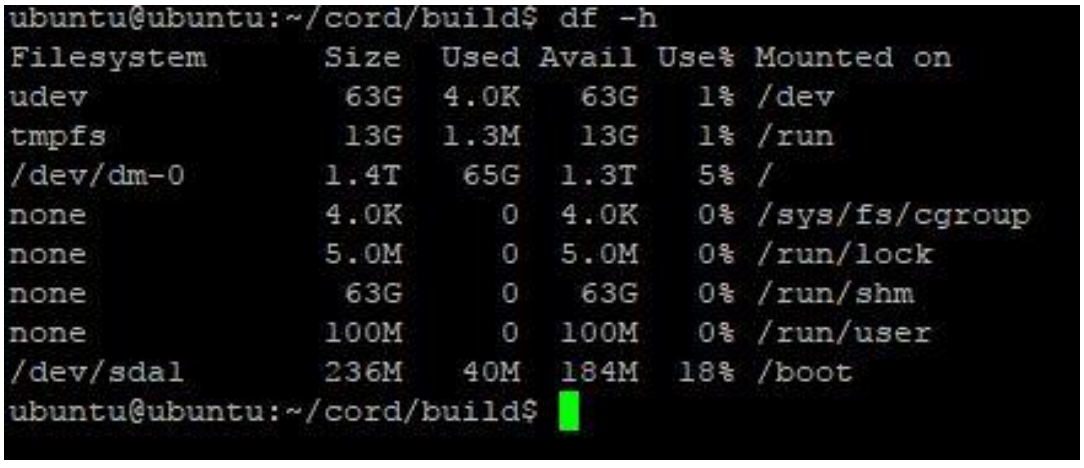


Fig. 3.22 Disk memory in UPC CORD

Here, the main partition is “/dev/dm-o”, which has the following characteristics:

-Size: 1.4 TB
-Used: 65 GB
-Availabe: 1.3 TB
-Use: 5%

3.3.3 Conclusions

If we observe the different situations and the samples obtained, we will realize that there is a moderate CPU consumption increase when a new client is being added. Moreover, the addition of a new subscriber does not suppose a high increase in the disk memory.

We have also observed an unusual behaviour. In Cloudlab, almost the 100% of the RAM is used (67 GB more or less), and it works perfectly. In the UPC physical server, the installation uses only 93 GB of 134 GB available. For this reason, we think that Cord in a Box can work properly in a system with a not very high capacity, adapting to existing conditions.

CHAPTER 4. CIAB WITH 3 COMPUTE NODES

4.1 Definition of the scenario

In order to make the system look more like the reference implementation, we have installed and tested Cord in a Box with three compute nodes.

Now, the system has the following structure inside the same physical/virtual machine:

- 1 dev node
- 1 head node
- 3 compute nodes

It is explained in depth in section 4.2. In Annex C the different ways to install Cord in a Box with three compute nodes are explained.

4.2 Explanation of the scenario

Once the previous steps have been followed, the situation shown in Fig. 4.1 is reached:

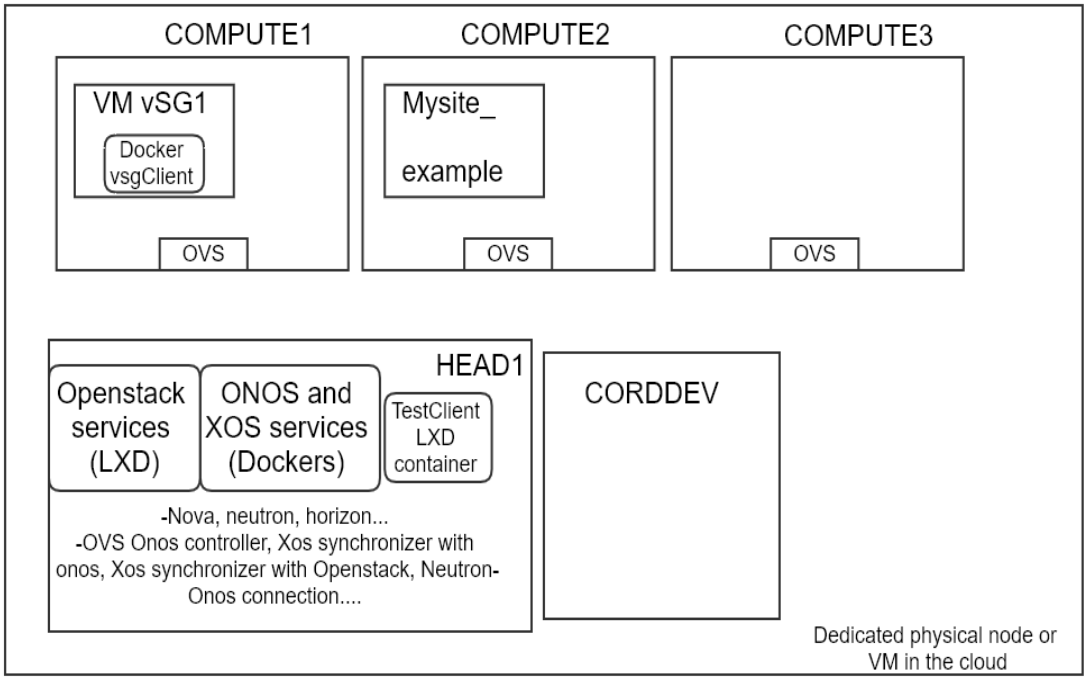


Fig. 4.1 High level scheme of CiaB with 3 compute nodes

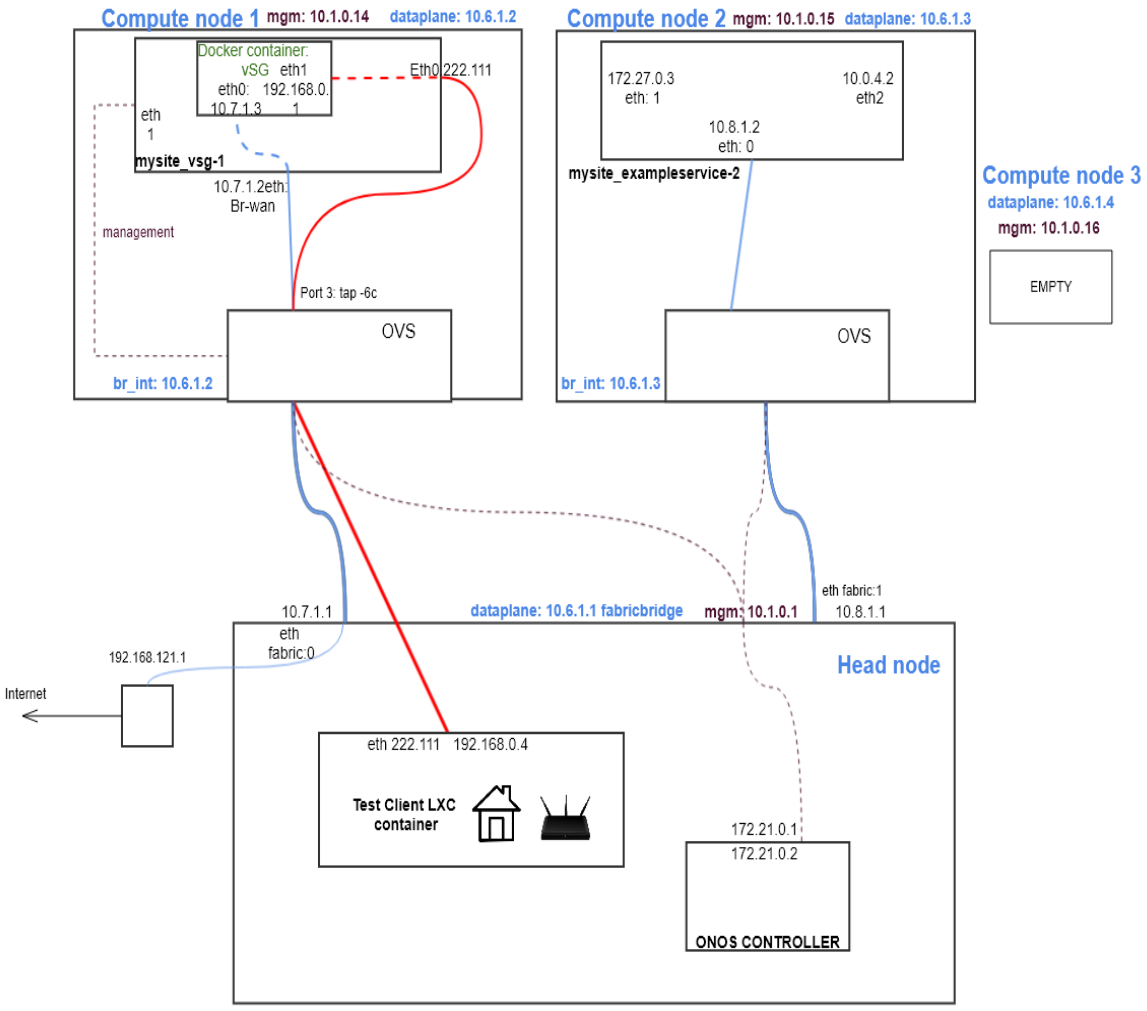


Fig. 4.2 Detailed scheme of CiaB with three compute nodes

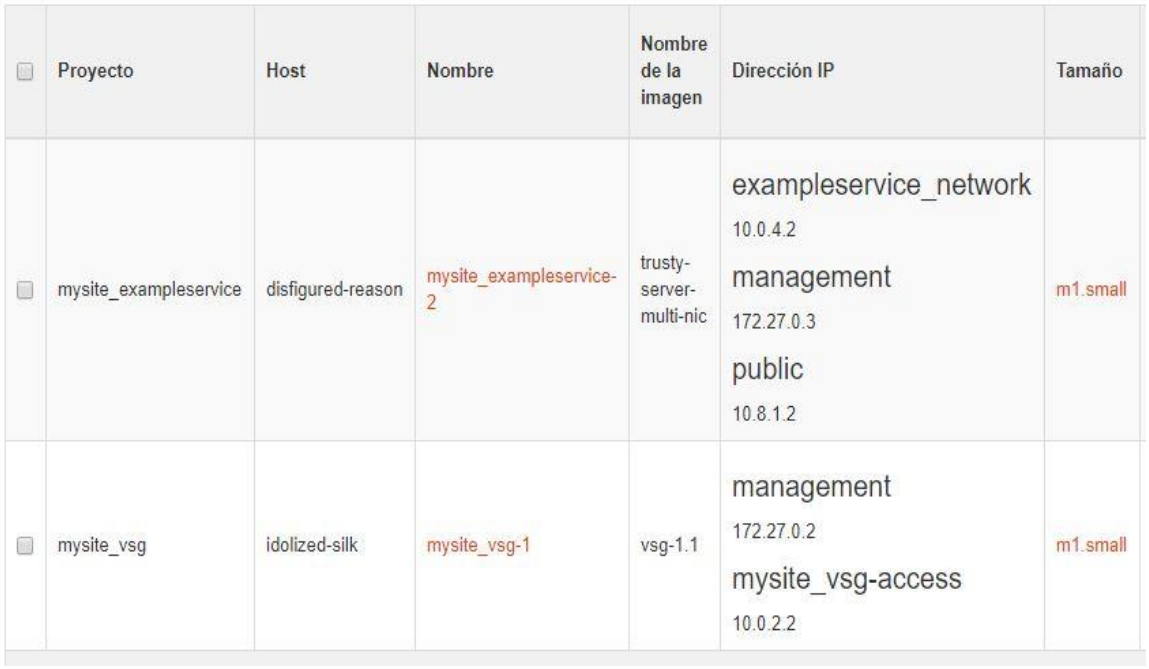
As Fig.4.2 shows, every compute node has an OVS controlled by ONOS controller in the head node, as can be seen in the UI of ONOS. Also, every virtual machine inside the different compute node is controlled by OpenStack Nova module, as can be seen in the OpenStack dashboard. We explain how to access to the different UIs in Annex B.

Devices (3 total)

	FRIENDLY NAME	DEVICE ID	MASTER	PORTS	VENDOR
✔	of:0000525400030791	of:0000525400030791	172.21.0.2	6	Nicira, Inc.
✔	of:00005254008db010	of:00005254008db010	172.21.0.2	5	Nicira, Inc.
✔	of:0000525400bad08f	of:0000525400bad08f	172.21.0.2	3	Nicira, Inc.

Fig. 4.3 ONOS UI

Fig 4.4 shows Openstack dashboard:



Proyecto	Host	Nombre	Nombre de la imagen	Dirección IP	Tamaño
mysite_exampleservice	disfigured-reason	mysite_exampleservice-2	trusty-server-multi-nic	exampleservice_network 10.0.4.2 management 172.27.0.3 public 10.8.1.2	m1.small
mysite_vsg	idolized-silk	mysite_vsg-1	vsg-1.1	management 172.27.0.2 mysite_vsg-access 10.0.2.2	m1.small

Fig. 4.4 OpenStack dashboard

4.3 Adding a subscriber

This section describes the behavior of the system when a new subscriber is added in a different VOLT.

When the subscriber is added in a different vOLT, the new VM is assigned to the node with the fewest number of instances. VMs are assigned to compute nodes using the LeastLoadedNodeScheduler() function inside XOS [19] and it is used in [20].

As aforementioned, the function LeastLoadedNodeScheduler() returns the node with the fewest number of instances. Then, XOS makes a query to the Openstack Nova API (passing as parameter the selected node), in order to make Nova creating the new virtual machine in the chosen node. The current situation is explained in Fig. 4.5.

As shown in Fig. 4.5, the new VM is assigned to compute 3 (because until now it was the node with the fewest number of instances), and the other computes remain identical.

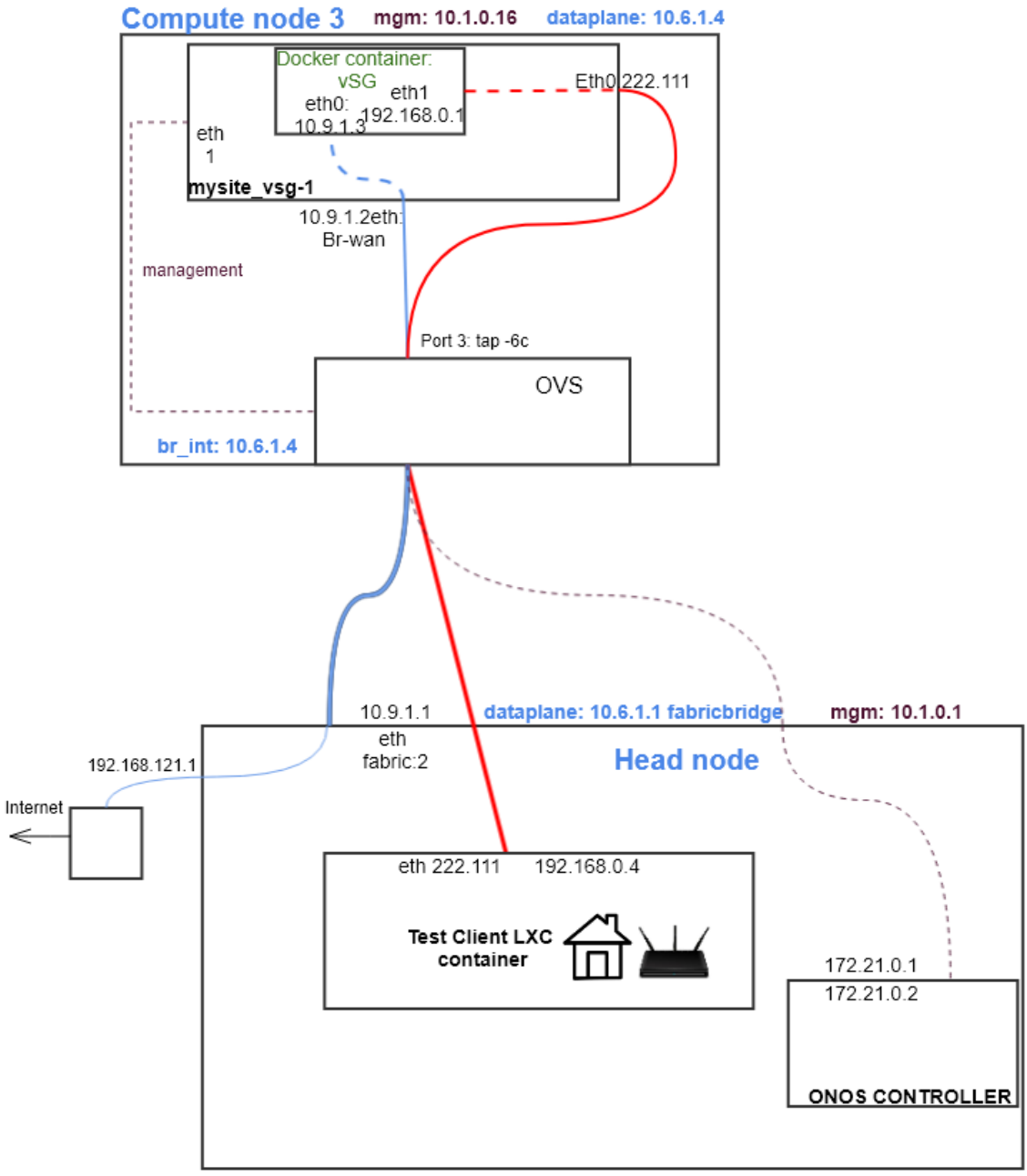


Fig. 4.5 Third compute node when new VOLT is added

4.4 Summary and conclusions

The previous sections have described CORD in detail. We know CORD develops several projects in parallel, and one of them is the system resources monitoring for its better efficiency. In Chapter 5, we propose an improvement of the energy efficiency of the Openstack module of CORD.

CHAPTER 5. IMPROVING OPENSTACK'S ENERGY EFFICIENCY IN CORD

5.1 Proposal - Introduction

As we have seen, CORD is a promising option for becoming the future central offices architecture of. The central offices expenses represent a very high percentage of the service providers' total expense. Datacenters consume 2.5% of the total electricity generated in the United States, and according to some articles, they will consume 20% of earth power in 2025 [21] [22]. For these reasons, we believe in the importance of energy efficiency in access networks.

We propose to implement the concepts developed by Ferran Diaz's Bachelor Thesis Degree [23] in CORD. In his project, Ferran Diaz proposes an improvement of the Openstack virtual machine allocation algorithm. Specifically, he changes Ceilometer and Nova components, in order to achieve:

- Creating virtual machines in the less-loaded node (not the node with less number of virtual machines, as CORD does, but the node with less CPU consumption).
- Migration when a node is overused.
- Migration when a node is underutilized.
- Deleting virtual machines when not used.

5.1.1 Proposal – Specification

We propose to make the necessary changes to CORD's Openstack based on Ferran's work, in order to achieve the next objectives:

- Live migration[21] when a node is underutilized.
- Live Migration when a node is overused.

- **Use case of Live Migration when a node is underutilized**

A service provider that only provides service in one country, or in different countries within the same time zone, knows that its central office will have more traffic in daytime hours, and it will have its nodes underused in night hours.

In this case, let's imagine that there is one reference implementation of CORD, with three compute nodes and different virtual machines and Dockers in every node. When night arrives, most of the residential subscribers go to sleep and the traffic decreases considerably.

[21] Live migration refers to the process of moving a running virtual machine or application between different physical machines without disconnecting the client or application. Memory, storage, and network connectivity are trnsferred from the original guest machine to the destination.

With the alerts we have programmed, the system detects automatically that traffic in one node has decreased below a scale, checks if there is a node to support the incorporation of all virtual machines existing in the other node, and it implements live migration to these virtual machines. Then, the underused node can be turned off.

- **Use case of Live Migration when a node is overused**

In the same case, when the subscribers wake up, traffic in the central office is increased. Now, the system detects automatically that the traffic in one node is above a scale. It checks if there is another node turned on to make the live migration to this node and if not, it turns on the node turned off the night before.

5.2 Steps to achieve it

First of all, it is important to say that, compared with Ferran's project, the situation is different. Openstack CORD components are dockerized, while, in Ferran's work, they were developed in a regular environment, with different applications and services, running in the same virtual machine. So, the steps required to reproduce the same scenario in CORD are the following:

1) Search for the component versions of Ferran's work and compare it with CORD versions of the same modules.

2) Creating a new Docker with RabbitMQ[22] in order to read the CORD queues.

3) Communicate with the Nova queue of RabbitMQ.

4) Making the necessary changes to Openstack CORD modules and Ferran's modules (also, it is important to remember that Ferran's work does not implement live migration, but only migration).

5.3 Experiments

Regarding the step 1 (component versions), there has been no problem. We have checked it and the versions are completely compatible.

We have not been able to fully perform this experiment because we have had many problems at step 2. We created and initializated a new docker with GO image [24]. In this docker, rabbitmq-dump-queue [25] library was installed.

Following its guide and applying Nova credentials, we run this command in order to dump the first 50 messages of the queue:

[22] RabbitMQ is an open-source message bróker that supports Streaming text oriented messaging protocol (STOMP), Message queuing telemetry transport (MQTT), and other protocols.

```
rabbitmq-dump-queue                                                    -
uri="amqp://nova:99PGT5ZMBryZhnwZh6N8kRY3Pw4LY9BPSfCxnLzYzfy3tJY3zKpkKnj6wKC4
pypx@rabbitmq-server.cord.lab:5671//openstack"       -queue="cert.nova-cloud-
controller-1"
```

Unfortunately, we got this error:

```
log: rabbit@10-1-0-12.log:
=ERROR REPORT==== 29-Apr-2018::15:50:17 ===
error on AMQP connection <0.19942.17>: {ssl_upgrade_error,
                                        {tls_alert,"record        overflow"}}
(unknown POSIX error)=ERROR
```

After studying it, we realized that the issue was related with certificate problem in RabbitMQ server. The configuration file of RabbitMQ was modified, allowing users to access without certificate:

```
{fail_if_no_peer_cert,false}
```

After that, RabbitMQ server was restarted, but the same error was still appearing.

At this point, and without knowing how to continue advancing, we have directly changed the CORD modules (by directly accessing the configuration file of each module, and modifying the necessary code lines), without reading the RabbitMQ queues. It was very difficult to work like this, because we could not see many environment variables. Nevertheless, we made the necessary changes in the necessary modules (chaning the code lines that Ferran explains in his project).

When we finished making the changes in a specific module, we compiled the code by restarting the service. For example, when we finished the changes in Nova, we restarted the Nova service, in order to force Nova to read its own configuration file. We did this for all the modules that Ferran mentioned.

Unfortunately, when we completed the changes in all the modules and restarted the services, nothing happened, and the behavior of the system did not change.

We think that this happened because the different services do not read their configuration from the typical configuration file (.conf). This theory is confirmed by the fact that when we changed the RabbitMQ server configuration file, after restarting the server and force to read its configuration file, the same certificate error kept appearing.

CHAPTER 6. CONCLUSIONS AND FUTURE LINES

6.1 Conclusions

The main goal of this project was to describe CORD in detail, creating as much documentation as possible, with detailed schemes, so that reading this project would be enough to understand CORD in depth. Another objective was to apply an algorithm to improve the Openstack energy efficiency.

To reach the first goal, we installed Cord in a Box 5.0, and Cord in a Box 5.0 with three compute nodes. It has been done in a physical server at Universitat Politècnica de Catalunya, and in a virtual machine in Cloudlab.

The goal of the first part of the project was, at a high level, to understand what CORD was and its purpouse, comparing it with legacy central offices architecture. Once understood, we installed Cord in a Box 5.0, studying it in depth, in order to create the first schemes and describe the system with more detail. We also applied a patch [26] from Andy Bavier's and through that, three compute nodes were added to the system, so we made our installation even more like real CORD. It has also been seen what happens if a subscriptor is added to the system. This part includes Chapter 2, 3 and 4.

After explaining CORD in detail, with all its components and created all the documentation and schemes necessary, we proceeded to describe and try to apply the improved algorithm in Openstack.

Although the first part has been successfully achieved, the second part has not been possible to be fully performed. When we have tried to apply the algorithm, it has failed. We have not been able to read the RabbitMQ queues because of a certificate problem. Furthermore, we have not found the way to force the different services to read their configuration file once the changes have been made.

During many years, the Central Office was being prepared as the growth of the Global Internet progressed, without time to form a really efficient architecture. CORD not only improves the existing Central Office, but it is prepared to the new era: the massive data era (Big data era). Any person, company or group can collaborate and improve it. This fact helps to continuous improvement and quick adaption, much needed because of the drastic and fast changes of today. We can say that CORD is very dynamic: one year ago, the most recent version was CORD 3.0. Today, there is CORD 5.0, and CORD 6.0 is about to be launched. In order to improve the system, many changes are being made but, definitely and in a very short time, CORD will be a very stable system for the Central Offices. The necessary information to contribute with CORD is in [27].

To conclude, we can say that CORD is a very scalable and efficiency system, and improves, by far, legacy central offices architecture. Furthermore, CORD

considerably reduces the operational and maintenance expenses of the Central Office, being this a great advange with respect to what is currently used.

6.2 Future Lines

Obviously, CORD will take some time to be present at Central Offices but, in our opinion, it will become the main architecture for Central Offices.

In this project, it has not been possible to succeed with one of the many objectives. It would be very interesting to complete it, because it will help to improve the energy efficiency of all the system. Likewise, any improvement in other area will be good.

In the Introduction section we have said that there are seven projects under CORD:

- Residential CORD (R-CORD)
- Enterprise CORD (E-CORD)
- Mobile CORD (M-CORD)
- Analytics for CORD (A-CORD)
- Trellis: CORD network infrastructure
- XOS and the CORD controller
- VOLTHA: Virtual OLT Hardware Abstraction

This work has only dealt with R-CORD (and it can be expanded), so there are six more projects to continue investigating. For example, there a lot of people interested about Mobile CORD. If you are interested in M-CORD (which expands CORD for 5G networks), you can read about Newtork Slicing, which is a new virtual network architecture.

Analytics for CORD can also be vey interesting to study. What we propose in Chapter 5 of this project is related with Analytics CORD.

6.3 Considerations about sustainability

There many important repercussions. As it has been said, CORD reduces the expenses of network providers. This fact could result in a reduction of the Internet subscription price.

In addition, home routers lifecycle could be extended (because through CORD, the complexity of a typical home router is reduced). Consequently, pollutant emissions would also be reduced. If Ferran’s algorithm is finally introduced in CORD, it would suppose important reductions of energy consumption.

To conlude, CORD has social, environmental and economics benefits. Because of these reasons is, without any doubt, a sustainable project in the short, medium and long term.

BIBLIOGRAPHY

1. Lima, Joao Marques. Data-economy. [Online] 10 12, 2017. [Cited: 02 05, 2018]. https://data-economy.com/data-centres-world-will-consume-1-5-earths-power-2025/

2. Prete, Luca. Wiki Cord. [Online] 09 14, 2017: https://wiki.opencord.org/pages/viewpage.action?pageId=2557405

3. CORD Wiki. [Online]. https://wiki.opencord.org/

4. CORD Guide. [Online]. https://guide.opencord.org/

5. ONF. Opennetworking. [Online] 2018. https://www.opennetworking.org/sdn-definition/

6. Opennetworking, SDN definition. https://www.opennetworking.org/sdn-definition/

7. ONOS project. [Online] 2018. https://onosproject.org/

8. Nagralawala, Nakul. Difference between SDN and NFV. [Online] 12 10, 2016. https://www.netmanias.com/en/post/blog/11022/sdn-nfv/difference-between-sdn-and-nfv-which-one-is-better

9. Openstack. Openstack. [Online] 2016. https://www.openstack.org/

10. Arena, Unix. Openstack Conceptual. [Online]: https://www.unixarena.com/wp-content/uploads/2015/08/Openstack-Conceptual-UnixArena1.jpg

11. L.Peterson, A. Al-Shabibi, T.Anshutz, S.Baker, A.Bavier, S.Das, J.Hart, G.Palukar and W.Snow, “Central Office Re-architected as a Datacenter.” IEEE Communications Magazine, October 2016. Pages 96-101. [Accessed: 10-February-2018] http://ieeexplore.ieee.org.recursos.biblioteca.upc.edu/stamp/stamp.jsp?tp=&arnumber=7588276

12. CORD. [Online] 2016. https://opencord.org/

13. Peterson, Larry. CORD reference implementation. [Online] 01 11, 2017. https://wiki.opencord.org/display/CORD/CORD+Reference+Implementation

14. Docker. https://www.docker.com/

15. XOS, orchestrator. http://guide.xosproject.org/

16. Open vSwitch. http://www.openvswitch.org/

17. Cord in a Box. Guide installation. [Online] 2017. [Cited: 10 25, 2017] https://guide.opencord.org/cord-5.0/install_virtual.html

18. Vagrant. https://www.vagrantup.com/

19. Smbaker (github). XOS CORD Function. [Online] 01 09, 2019. https://github.com/opencord/xos/blob/master/xos/synchronizers/new_base/model_policies/model_policy_tenantwithcontainer.py#L41

20. CORD (github). XOS Function 2. [Online] 01 2018. https://github.com/opencord/xos/blob/master/xos/synchronizers/new_base/model_policies/model_policy_tenantwithcontainer.py#L271

21. Bawden, Tom. Data Centres to consume three times as much energy in next dexace, experts warn. [Online] 01 23, 2016. https://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html

22. Sverdlik, Yevgeniy. How Much Energy All US Data Centers Consume. [Online] 06 27, 2016. http://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume

23. Diaz, Ferran. Bachelor Degree Thesis, UPC, 2017. [Online] 2017. https://upcommons.upc.edu/browse?type=author&value=Diaz+Martinez%2C+Ferran

24. Golang image Docker Official repository. [Online] https://hub.docker.com/_/golang/

25. Dubek (github). Github RabbitMQ-dump-queue. [Online] https://github.com/dubek/rabbitmq-dump-queue

26. Bavier, Andy. Gerrit Patch. https://gerrit.opencord.org/#/c/8306/

27. CORD. CORD contribution. [Online] . https://opencord.org/contribute/

28. CORD forum developers group. https://groups.google.com/a/opencord.org/forum/#!forum/cord-dev

29. CORD forum discuss group. https://groups.google.com/a/opencord.org/forum/#!forum/cord-discuss

GLOSSARY

API	Application Programming Interface
BNG	Broadband Network Gateway
CDN	Content Delivery Network
CiaB	Cord in a Box
CPE	Customer Premises Equipment
DSLAM	Digital Subscriber Line Access Multiplexer
GPON	Gigabit Passive Optical Network
NFV	Network Function Virtualization
OFP	Open Flow Protocol
OLT	Optical Line Terminal
ONOS	Open Network Operating System
ONT	Optical Network Termination
ONU	Optical Network Unit
OVS	Open vSwitch
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
SDN	Software Defined Network
TDM	Time-division multiplexing
vBNG	Virtual BNG
VM	Virtual Machine
VNS	Virtualized Newtork Services
vOLT	Virtual OLT
vRouter	Virtual Router
vSG	Virtual Subscriber Gateway

ANNEXES

ANNEX A. Installation of basic *CiaB*

To run and complete the installation, we have done four steps:

Note: If there is an error during the installation, you could find the solution in annex D.

1-Configuring password-less sudo capabilty:

```
ubuntu@node: ~$ sudo visudo
```

Adding at the last line:

```
“Myusername” ALL=(ALL:ALL) ALL
“Myusername” ALL=NOPASSWD: ALL
```

And restarting ssh service.

2-Bootstraping the server running (if you want CiaB 4.0, change 5.0 for 4.0):

```
ubuntu@node: ~$  curl -o ~/cord-bootstrap.sh
https://raw.githubusercontent.com/opencord/cord/cord-5.0/scripts/cord-
bootstrap.sh
```

```
ubuntu@node: ~$ chmod +x cord-bootstrap.sh
ubuntu@node: ~$./cord-bootstrap.sh –v
```

3-Running the installation in a tmux session:

```
Tmux
ubuntu@node: ~$  cd ~/cord/build
make PODCONFIG=rcord-virtual.yml config
make -j4 build |& tee ~/build.out
```

4-Running end-to-end tests to create the first simulated subscriber.

```
cd ~/cord/build
make pod-test
```

ANNEX B. CiaB GUIs

At this point, 1 CORD POD is running with 1 subscriber and the following modules can be accessed:

-MAAS GUI:

`http://IPorNodeName:8080/MAAS/`
user: cord
password: cord/build/maas/passwords/maas_user.txt

-OpenStack GUI:

```
ubuntu@node: ~$ ssh -L 0.0.0.0:9999:10.1.0.11:80 head1
```
`http://IPorNodeName:9999/horizon/`
user: admin
password: cord/build/platform-install/cord_keystone_admin

-XOS GUI:

`http://IPorNodeName:8080/xos/`
user: xosadmin@opencord.org
password: head1: /opt/credentials/xosadmin@opencord.org

-ONOS GUI:

`http://IPorNodeName:8080/vtn/`
user: karaf
password: karaf
`vagrant@head1: ~$ ssh -p 8102 onos@onos-cord` or
`vagrant@head1: ~$ ssh -p 8102 onos@localhost` (from head1)
password: rocks

-KIBANA GUI:

`http:// IPorNodeName:8080/app/kibana`

ANNEX C. Installation of 3 computes *CiaB*

To carry out the installation successfully, the same prerequisites are needed as in CiaB with one compute.

At this moment (02/04/2018) it can be installed in the following ways:

-Through the master branch
-Through the version 5.0

We recommend to install it as explained in Annex C.2 (Version 5.0).

Note: If there is an error during the installation, you could find the solution in annex D.

C.1 Master branch installation

As explained in opencord guide [3], there are needed x steps:

1- Bootstrap the server:

```
curl -o ~/cord-bootstrap.sh
https://raw.githubusercontent.com/opencord/cord/master/scripts/cord-
bootstrap.sh
chmod +x cord-bootstrap.sh

./cord-bootstrap.sh –v
```

2- Build and deploy the software:

```
cd ~/cord/build
make PODCONFIG=rcord-virtual.yml config
make -j4 build |& tee ~/build.out
```

3- Bring up the second and third virtual compute nodes.

```
cd ~/cord/build
make compute2-up
make compute3-up
```

4- Bring up an emulated client by running end-to-end tests.

```
cd ~/cord/build
make pod-test
```

C.2 Installation of version 5.0

In this case, there is needed to use an Andy Bavier's patch hosted in Gerrit [26]. The following steps are needed to complete the installation:

1- Bootstrap the server:

```
curl -o ~/cord-bootstrap.sh
https://raw.githubusercontent.com/opencord/cord/cord-5.0/scripts/cord-
bootstrap.sh
chmod +x cord-bootstrap.sh

./cord-bootstrap.sh –v –p cord:8306
```

2- Build and deploy the software:

```
cd ~/cord/build
make PODCONFIG=rcord-virtual.yml config
make -j4 build |& tee ~/build.out
```

3- Bring up the second and third virtual compute nodes.

```
cd ~/cord/build
make milestones/compute2-up
make milestones/compute3-up
```

4- Bring up an emulated client by running end-to-end tests.

```
cd ~/cord/build
make pod-test
```

ANNEX D. Possible mistakes during installation

Recommendations:

-Run the “Make” command in a “tmux” session. Otherwise, the system could fail because of a timeout error.

-Run the installation in a Physical server. Make sure you accomplish all the prerequisites.

-If your installation doesn not run, request an account to cloudlab and Larry Peterson, maybe you can get an account and run the installation in a virtual machine.

Error type 1:

-TASK [copy-cord: Copy (sync) the cord directory to head node]. Failed to transfer file to X. No such file or directory X.

Solution:

-Downgrade ansible from 2.5.3 to 2.5.2 (this step requires upgrade python setuptools):

```
sudo apt-get remove ansible
sudo apt-get remove python-setuptools
wget https://bootstrap.pypa.io/get-pip.py
sudo –H python get-pip.py
sudo –H pip install –U pip setuptools
sudo pip install ansible==2.5.2
```

Error type 2:

-Any TASK related with Openstack.

Solution:

-The installation will rarely work in a virtual machine with Openstack environment.
-If you have not run the installation in a “tmux” session, maybe it can fail here.
-If the previous cases do not correspond with yours, ask in CORD forums in [28] and [29].