TESIS

TÍTULO:

VIRTUALIZACIÓN EN REDES DEFINIDAS POR SOFTWARE

Alumno:

Mariano Nicolás Osaba
Ingeniero en Electrónica

Carrera:

Maestría en Ingeniería de las Telecomunicaciones

Instituto Tecnológico de Buenos Aires

Director de Tesis:
MSc. Ing. Alejandro Juan Manuel Repetto

2016

Abstract

En la última década surgió un nuevo paradigma para las redes de datos que surge a partir de un trabajo de investigación de la Universidad de Stanford a partir del cual se introduce el concepto de Software Defined Networks (SDN). El mismo promete soluciones a algunas de las limitaciones de las redes actuales, como ser: la baja tasa de innovación en los dispositivos, la complejidad para su administración y la falta de flexibilidad para adaptarse a los cambios.

El presente trabajo aborda la intersección entre SDN y virtualización de las redes relevando y definiendo atributos requeridos para soluciones de virtualización en general y resultados de dos implementaciones en particular (OVX y FV), caracterizados a través de la realización de pruebas en entornos simulados, y proponiendo una metodología para la comparación y selección.

La definición de atributos y la metodología de caracterización propuesta no están acotadas sólo a las soluciones abordadas, sino que sirven de framework para la futura evaluación en otros contextos.

Surge del trabajo realizado que las soluciones de virtualización en el contexto de SDN presentan ventajas respecto a las correspondientes a arquitecturas tradicionales, entre las cuales se destacan: la agilidad de los despliegues a través de la automatización; la capacidad para una gestión centralizada, facilitando la administración; la posibilidad de desarrollo continuo de nuevas funcionalidades sin cambiar el HW; la facilidad para la creación de ambientes multitenancy; la provisión de IaaS y los procesos de migración.

INDICE

LISTADO DE FIGURAS

LISTADO DE ABREVIATURAS

AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BPDU	Bridge Protocol Data Units
CAPEX	Capital Expenditure
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
ETH	Ethernet
GRE	Generic Routing Encapsulation
HW	Hardware
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IT	Information Technology
I/O	Input/Output
JSON	JavaScript Object Notation
LLDP	Link Layer Discovery Protocol
LUN	Logical Unit Number
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
MSTP	Multiple Spanning Tree Protocol
NFV	Network Function Virtualization
NIC	Network Interface Card
NOS	Network Operating System
ODL	OpenDayLight
OH	Overhead
ONF	Open Networking Foundation
OPEX	Operating Expenditure
OS	Operating System

OSI	Open System Interconnection
OVS	Open vSwitch
PCP	Priority Code Point
PVST	Per VLAN Spanning Tree
QoS	Quality Of Service
REP	Resilient Ethernet Protocol
RFC	Request For Comments
RSTP	Rapid Spanning Tree Protocol
SDDC	Software Defined Datacenter
SDE	Software Defined Environment
SDN	Software Defined Networks
SDx	Software Defined Everything
SSH	Secure Shell
STP	Spanning Tree Protocol
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	Virtual Machine Monitor
VN	Virtual Network
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
vSDN	Virtual Software Defined Network
WAN	Wide Area Network
WDM	Wavelength Division Multiplexing

I.INTRODUCCIÓN

1. INTRODUCCIÓN

Las redes de datos tradicionales en la actualidad presentan limitaciones que no permiten que su velocidad de evolución sea comparable con la relativa a los servicios que soportan. En los últimos años, ha surgido un nuevo paradigma de *networking* que se está imponiendo en el mercado y que supone un cambio de arquitectura y, en consecuencia, de funcionamiento de las redes. Éste se denomina *Software Defined Networking (SDN),* el cual impone la separación del plano de datos respecto del de control y la definición de interfaces abiertas, posibilitando así el desarrollo y evolución de las redes, en pos de lograr mayor agilidad y flexibilidad.

De manera complementaria, la implementación de SDN supone un sustrato para la virtualización de las redes. Este es un tema de actualidad y de interés por parte de los ámbitos académicos y la industria, representando un caso de uso que puede resultar de gran impacto en la misma.

El presente trabajo aborda este área, específicamente la intersección entre SDN y virtualización de las redes, contexto en el cual se propone el estudio de algunas tecnologías representativas bajo este dominio. Se relevarán y definirán atributos requeridos para este tipo de soluciones; se caracterizarán a través de la implementación de pruebas en entornos simulados; y se propondrá una metodología para la comparación y selección.

2. OBJETIVOS

A. OBJETIVO GENERAL

Profundizar el conocimiento sobre virtualización en el contexto de SDN; definir atributos y una herramienta de comparación que permitan determinar aquellas soluciones con potencial de ser desplegadas en un futuro en redes productivas.

B. OBJETIVOS ESPECÍFICOS

1) Definir atributos relevantes que permitan la caracterización de las soluciones de virtualización de las redes.
2) Relevar, estudiar y caracterizar propuestas de virtualización representativas de redes SDN.
3) Consolidar los atributos correspondientes a cada una de las estrategias analizadas a través de su implementación en entornos de simulación.
4) Realizar una comparación objetiva de las estrategias de virtualización analizadas.
5) Generar un framework que sirva para evaluar de manera general estrategias de virtualización en redes SDN.

3. ALCANCE

Este trabajo se va a limitar al análisis de soluciones de virtualización de las redes en el contexto de *datacenters* corporativos. Para ello, se analizarán y simularán dos tecnologías de virtualización en SDN, representativas en el contexto propuesto (FlowVisor y OpenVirtex).

De esta simulación y del relevamiento bibliográfico se obtendrán las características propias de la virtualización de redes, a partir de lo cual se propondrá una metodología de comparación y se expondrán los resultados obtenidos.

4. METODOLOGÍA

A. MARCO TEÓRICO Y ESTADO DEL ARTE

La definición del marco teórico y el estado del arte es el punto de partida del proyecto de tesis. A través este relevamiento, se buscará identificar el estado actual de evolución de las tecnologías que son objeto del presente trabajo. Ello permitirá la profundización de los conocimientos en la materia y la definición específica de los lineamientos de la investigación.

Esta etapa se caracterizará por el relevamiento bibliográfico, búsqueda de antecedentes, estándares y lectura de dicho material.

B. DESARROLLO DE LOS OBJETIVOS

Para la resolución de la problemática planteada se utilizarán las mismas herramientas expuestas en el apartado anterior. Se adicionará a dicha metodología, la realización de entrevistas de campo a referentes del área y/o a representantes de la industria. También se utilizarán herramientas de simulación y/o software libre que permitan la realización de pruebas prácticas de las tecnologías abordadas.

C. FUENTES DE INFORMACIÓN

1) Bibliografía de actualidad
2) Sitios Web
3) Organismos internacionales de estandarización
4) Información y hojas de datos de fabricantes
5) Libros, tesis y/o otras publicaciones científicas
6) Información libre publicada por operadores/empresas
7) Entrevistas a especialistas y/o referentes
8) Experiencias de proveedores de equipamiento de telecomunicaciones
9) Comunidades de desarrollo de aplicaciones de uso libre/código abierto
10) Participación en eventos y/o exposiciones relacionadas con la temática

II.MARCO TEÓRICO

1. SDN

A. ANTECEDENTES

Las redes de datos como las conocemos en la actualidad surgieron a partir de los requerimientos de las primeras redes militares y de la aparición de Internet [1]. Ésta última fue concebida como una red de redes cuyo funcionamiento se basa en la conmutación de paquetes. En ella cada componente debió funcionar de manera inteligente, es decir con capacidad inherente para la toma de decisiones autónomas que aseguren la transmisión de información de extremo a extremo. En este proceso, los componentes involucrados en la comunicación, no poseían una visión integral de la red.

Desde su concepción, los componentes físicos requeridos para el funcionamiento de las redes nacieron como estructuras monolíticas con hardware, software e interfaces propietarias. Entre otras, dicha característica generó que estos dispositivos evolucionen de manera lenta con respecto a otras tecnologías, como por ejemplo servidores y soluciones de almacenamiento. La consecuencia directa de esta situación se ve reflejada en la incapacidad de las redes para soportar los requerimientos actuales de "*time to market*" en la provisión de nuevos servicios, dificultad en su gestión, complejidad y elevados costos de operación, entre otros [2].

Si hacemos una analogía en lo que respecta a la evolución de las soluciones de cómputo (Figura 1), los dispositivos de red en la actualidad se encuentran en una situación similar a la de los *mainframe* en los años 80'[3].

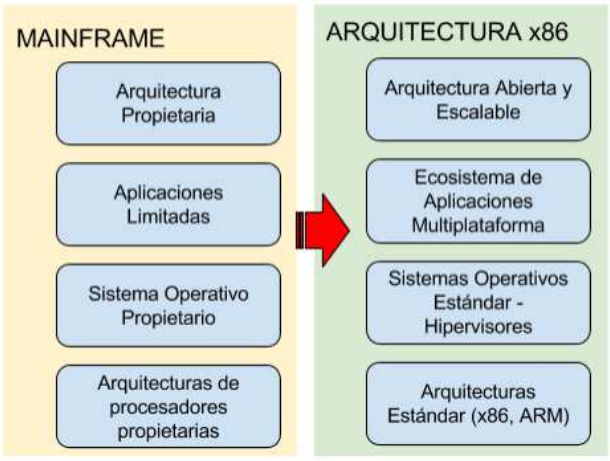


Figura 1 - Evolución de arquitectura de soluciones de cómputo

Las tecnologías móviles, la continua interacción de los usuarios con contenido en línea, las nuevas tecnologías de virtualización de servidores y el advenimiento de servicios en la nube, obligan a realizar una revisión de las premisas de diseño y funcionamiento de las actuales redes [4].

La última década ofrece un nuevo paradigma para las redes de datos que surge a partir de un trabajo de investigación de la Universidad de Stanford [5], donde se introduce el concepto de *Software Defined Networks* (SDN) que promete soluciones a las limitaciones de las redes actuales, agilizando el aprovisionamiento de nuevos servicios, facilitando la gestión a través de dispositivos físicos sencillos, introduciendo protocolos abiertos que posibilitan la innovación e incorporando componentes de software que permiten la abstracción de funcionalidades implementadas previamente a través de hardware.

B. REDES DE DATOS TRADICIONALES

Se presentan algunos de los problemas asociados al diseño y arquitectura de las redes de datos tradicionales, los cuales servirán de fundamento para justificar el análisis de un nuevo paradigma de *networking*, en este caso SDN.

De esta manera, se buscan consolidar las debilidades asociadas a las arquitecturas actuales, lo cual permite comprender las ventajas de estas nuevas tecnologías de redes en la prestación de servicios de valor agregado.

Se comienza analizando algunas de las premisas de diseño que denotan la complejidad por la que están transitando las redes en la actualidad. Estas son, su característica distribuida y la naturaleza monolítica de sus componentes, entre otras.

1) Característica Distribuida

J. Postel fue uno de los pioneros en lo que respecta a la estandarización de internet. El RFC (Request For Comments) 791 [6] expone sus características de diseño y funcionamiento.

Este estándar define que los componentes de la red, utilizan las direcciones de destino incluidas en las cabeceras de los datagramas para transmitir los mismos hacia su

destinatario, en un proceso denominado ruteo. El modelo de operación para ello implica que en cada *host* y *gateway* vinculado en una comunicación a través de internet reside un módulo que permite dicha funcionalidad. Estos módulos incluyen reglas comunes para interpretar los campos de dirección y poder así tomar las decisiones de encaminamiento, entre otras. Esta última función resulta fundamental, puesto que cada datagrama se trata de manera independiente. El diseño original sobre el cual se hace referencia, especifica que en Internet no hay conexiones o circuitos lógicos entre el origen y el destino, sino que la comunicación se basa en procedimientos de ruteo por cada datagrama en forma individual. De esta manera se puede apreciar con claridad que el diseño de la red Internet desde sus comienzos fue concebida con *inteligencia distribuida*.

Si bien en el párrafo anterior se hace referencia a este estándar de los orígenes de internet, las redes en la actualidad siguen funcionando con la premisa de diseño expuesta. Es así que hoy las redes de datos están compuestas por switches y routers, que posibilitan las comunicaciones entre clientes y servidores físicos o virtuales, conformando redes complejas y difíciles de administrar [7]. Para lograr las funcionalidades según diseño, los operadores requieren configurar cada dispositivo en forma individual, utilizando comandos específicos que en la mayoría de los casos son propietarios del fabricante [3].

La característica distribuida de las redes actuales se basa en un diseño descentralizado en el cual la lógica de control y la función de distribución de paquetes (ruteo/forwarding) está embebido en cada uno de los componentes de la red. Es así que cada router o switch soporta una serie de protocolos distribuidos que facilitan la toma de decisión en el direccionamiento de paquetes a lo largo de la red [8]. Ejemplos de dichos protocolos pueden ser OSPF, IS-IS, EIGRP, STP, entre otros (Figura 2)

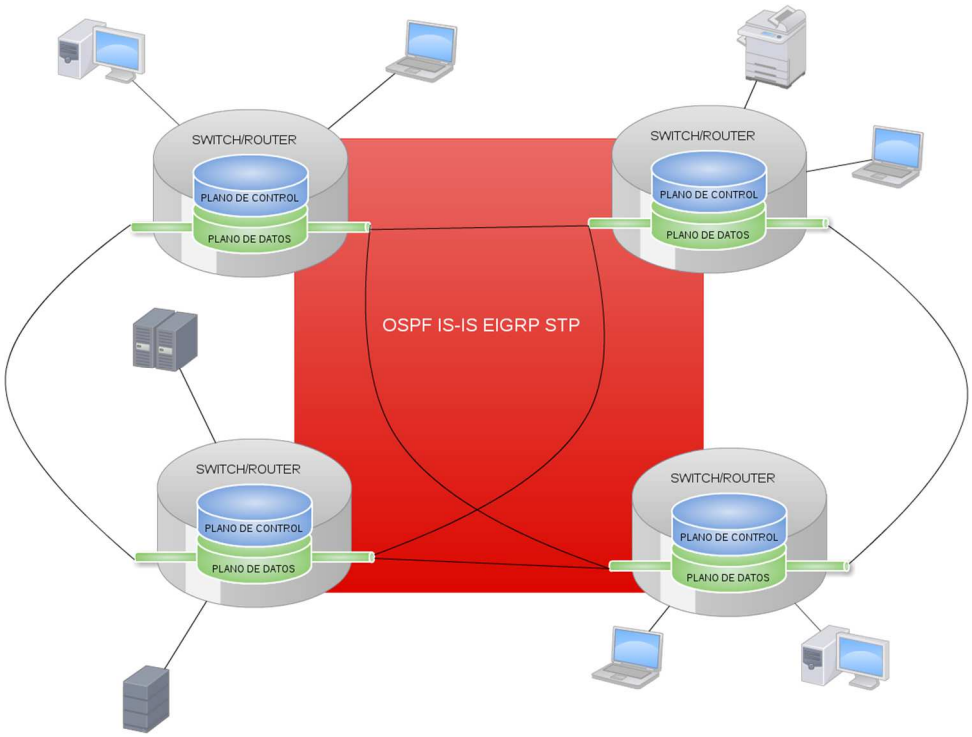


Figura 2-Red de datos tradicional

2) Naturaleza monolítica de los componentes de la red

Los equipos que posibilitan el encaminamiento de paquetes a través de las redes en la actualidad (Routers/Switches) están diseñados con una arquitectura del tipo monolítica [4]. Esta caracterización responde a que los mismos están diseñados de acuerdo al funcionamiento tradicional de las redes de datos, es decir incorporando en un mismo dispositivo físico el plano de datos, el de control y gestión, para así posibilitar el ruteo de individual de los paquetes entrantes (Figura 3).

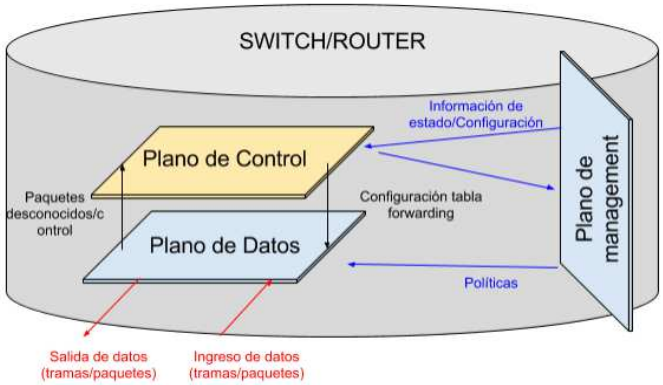


Figura 3 - Arquitectura monolítica de switches y routers

En el caso de un dispositivo de capa 2 (switch) del modelo OSI (Open System Interconnection), el plano de datos o forwarding, está constituido por los puertos físicos a través de los cuales se reciben y se transmiten las tramas de datos. Su principal función es la de encaminar éstas hacia los puertos de salida, utilizando para ello la información contenida en la tabla de forwarding embebida. Si la información de cabecera de una trama entrante, es encontrada en dicha tabla, esta puede ser susceptible de modificaciones para luego ser transmitida a través del puerto correspondiente, sin intervención de los otros planos. Esta situación no sucede en todos los casos. El ejemplo más claro, es cuando la información de la cabecera no se encuentra aún mapeada en la tabla. Es en este caso, es necesaria la intervención del plano de control, cuya principal responsabilidad es la de mantener actualizada la información de forwarding, de manera tal de que el plano de datos pueda retransmitir las tramas sin su intervención. Para ello cuenta con una serie de protocolos de control que permiten modificar dicha información y que tienen la responsabilidad de mantenerla actualizada de acuerdo a la topología actual de la red. Por último, el plano de gestión o *management* tiene como función principal permitir la administración y configuración de dichos dispositivos de red.

La integración vertical expuesta por parte de los planos de datos, de control y de gestión, constituye el diseño monolítico sobre el cual funcionan los componentes de una red de datos en la actualidad.

3) Problemas de las redes de datos tradicionales

Se introducen algunos de los problemas por los que transitan las redes de datos en la actualidad, los cuales son en parte consecuencia de las características de diseño introducidas en el apartado 1) y 2) del presente capítulo.

i. *Baja tasa de innovación en lo que respecta a equipamiento de networking*

En la actualidad, son los incumbentes del mercado de *networking* los que poseen total control de la industria en relación a investigación, desarrollo e innovación en la materia.

Durante las dos últimas décadas, a pesar de las grandes inversiones en investigación y desarrollo por parte de los fabricantes de equipamiento, los avances tecnológicos

resultaron mínimos debido a los altos costos y a los largos períodos que conllevan los nuevos desarrollos en el área. Esto se da principalmente porque el diseño y la arquitectura del equipamiento de networking tradicional es propietario y cerrado. Se limitan así las posibilidades de desarrollo sólo a aquellos que tienen acceso a las interfaces e información de las mismos, mayoritariamente los propios *vendors*.

Por otro lado, es importante destacar que los altos costos de adquisición de equipamiento, constituyen en la actualidad una gran barrera para el ingreso al mercado de nueva competencia. Ambos factores resultan en un limitante en la velocidad de desarrollo, que hoy en día no alcanza a satisfacer las demandas y evolución del mercado, generando además dependencia en los incumbentes.

En conclusión, en lo que a innovación se refiere, las tecnologías de *networking* contrastan con otras en las cuales se ha trabajado en comunidades abiertas de desarrollo, vinculando la industria, las Universidades y esfuerzos particulares logrando avances sustanciales y minimizando así los costos. Ejemplos de ello son, Linux en lo que respecta a sistemas operativos, KVM en lo que respecta a virtualización y MySQL en lo relativo a base de datos. Muchos de ellos convertidos hoy, en estándares de facto en la industria de IT.

ii. *Complejidad y crecimiento de las redes actuales debido a la evolución en los servicios*

El crecimiento en tamaño, complejidad y la aparición de nuevos servicios en las redes de datos, han motivado la generación de nuevos protocolos y/o evolución de los existentes, para posibilitar la transmisión de datos extremo a extremo.

Desde el punto de vista de hardware, esto se visualiza en la evolución de *transparent bridges* a switches, routers y por último switches de capa 3.

En paralelo, se han debido incrementar los recursos de memoria y procesamiento de los mismos para poder soportar el creciente número de protocolos. La consecuencia de esta evolución se puede apreciar en los equipos de networking que hoy en día, deben soportar miles de RFCs y estándares de IEEE para funcionar en las redes actuales (Figura 4). La necesidad de generación de nuevos protocolos y/o evolución de los existentes para

satisfacer las demandas del mercado, se puede ejemplificar a través de la evolución de STP (Spanning Tree Protocol).

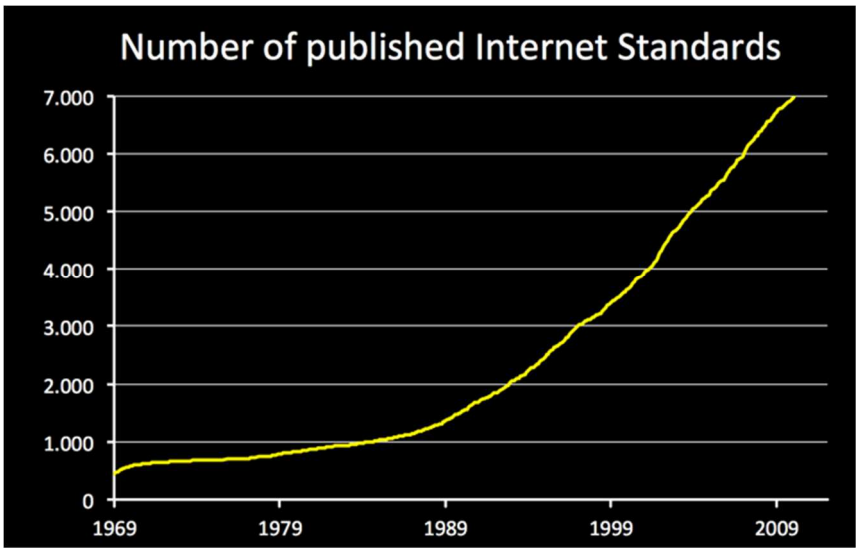


Figura 4 - Software Defined Networks and the Maturing of the Internet (McKeownm, 2014)

STP permite la eliminación de bucles mediante el control del estado de los enlaces de la red. Para ello se utiliza un tipo especial de tramas denominados BPDUs (Bridge Protocol Data Units). En el caso de que un camino falle, STP deberá volver a configurar la misma. La convergencia ocurre cuando pasa un intervalo de tiempo que permita asumir que todos los switches independientes han configurado sus puertos adecuadamente asegurando la inexistencia de *loops*, para así poder volver a cursar tráfico. La versión original (IEEE 802.1D año 1990) supone un tiempo de convergencia de 50 segundos. Ello implica que, durante este intervalo, todas las interfaces se mantienen en estado bloqueado manteniendo la red no operable. Hoy en día resultan inviables tiempos de interrupción como los expresados, por lo cual dicho protocolo ha sufrido modificaciones.

Se debe considerar para justificar dicha evolución, la convergencia de las redes en lo que respecta a voz, datos y video. Este tipo de servicios demandan tiempos de respuesta mucho más exigentes que no superan algunos mseg. Además, STP en su versión original no soporta VLANs (Virtual Local Area Network) lo cual representa otra limitación. Lo anteriormente expuesto culmina con la generación de nuevos protocolos estandarizados y otros propietarios:

- Rapid Spanning Tree Protocol (RSTP) estandarizado en IEEE 802.1w.
- Multiple Spanning Tree Protocol (MSTP) estandarizado en IEEE 802.1s.
- Per VLAN Spanning Tree (PVST) propietario de CISCO.
- Otros: PVST+, RPVST y Resilient Ethernet Protocol (REP).

En muchos casos, la velocidad de respuesta de los organismos de estandarización, concluye en la generación de protocolos propietarios que luego resultan estándares de facto en el mercado. Un ejemplo de ello es PVST. Ello complejiza aún más el escenario.

El funcionamiento básico de los switches (MAC learning) no ha sufrido cambios desde sus orígenes. En contraposición, los servicios prestados por las redes de datos han evolucionado de manera vertiginosa manifestándose en la constante evolución de los protocolos de comunicaciones. El equipamiento de networking actual no puede evolucionar siguiendo las demandas requeridas por el mercado tal como se expresó en i. Baja tasa de innovación en lo que respecta a equipamiento de networking.

Por otro lado, la generación de un nuevo protocolo implica que todos los dispositivos en una red deben ser actualizados debido a lo expuesto en 1) Característica Distribuida. Este mismo escenario se presenta al momento de agregar o mover un dispositivo. En ambos casos representa un trabajo manual e individual por cada equipo por parte del administrador.

Lo expuesto en este apartado da cuenta de la complejidad que están transitando las redes de datos tradicionales. Ello genera que las mismas sean muy estáticas con una consecuencia directa en las posibilidades de evolución, imposibilitando su adaptación a las necesidades dinámicas por parte de los usuarios y a la provisión de nuevos servicios de valor agregado.

iii. *Costos crecientes de adquisición, implementación y administración de equipamiento de networking*

La situación expuesta en el apartado anterior, genera un escenario en el cual los equipos de comunicaciones se están volviendo muy costosos.

Esto se debe en parte a que cada switch que conforma la red de datos debe incorporar una instancia del software asociado a la funcionalidad del plano de control que posibilita

su funcionamiento autónomo. A su vez, éste se ha ido complejizando a partir de la incorporación líneas de código en respuesta a la evolución e incremento en el número de protocolos. Ello genera además una consecuencia directa en los requerimientos de memoria y CPU (Central Processing Unit).

En una red con inteligencia distribuida, la incorporación de dicho software en cada equipo de red, es un aspecto fundamental ya que permite el encaminamiento de paquetes al destino correspondiente a lo largo de la red.

Por otro lado, la complejidad tiene su consecuencia en la administración de las redes, lo cual genera un escenario de costos elevados que repercuten de manera directa en el OPEX (Operating Expenditure) de las organizaciones.

iv. *Cambio de los patrones de tráfico*

Los diseños convencionales de redes en general, son construidos de manera jerárquica, es decir utilizando distintos niveles o *tiers* de switches ethernet que configuran una topología de tipo árbol (Figura 5).

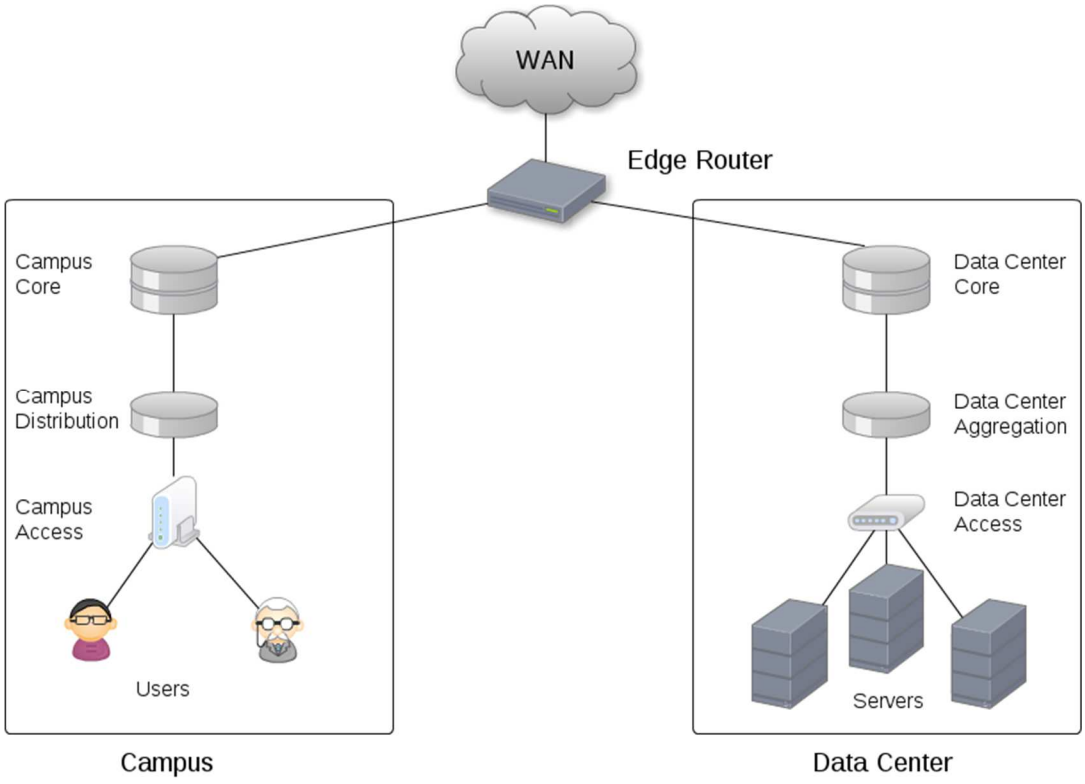


Figura 5 - Arquitectura de una red de datos

Este esquema resulta útil para el tráfico norte-sur característico en arquitecturas del tipo cliente servidor. Sin embargo, las aplicaciones actuales en muchos casos, requieren la interacción entre distintas bases de datos y servidores para generar una respuesta de cara al usuario. Esta situación genera en el centro de cómputos tráfico predominante del tipo este-oeste [4], generando un caso de uso para el cual no fueron diseñadas las redes actuales.

v. *Nuevos servicios en la nube: infraestructura como servicio*

La velocidad en la provisión de nuevos servicios ha evolucionado de manera superlativa en los últimos años. Los servicios en la nube y la posibilidad de adquirir IaaS (Infrastructure as a Service), han sido protagonistas en este sentido.

Sabemos que hoy fácilmente podemos adquirir soluciones de IaaS, que nos permiten el aprovisionamiento casi instantáneo de servidores, almacenamiento y bases de datos. Este es el caso de los servicios que ofrecen AWS (Amazon Web Services) y Microsoft Azzure entre otros, como referentes en el mercado.

En contraposición, las arquitecturas tradicionales de Networking, por su característica de diseño poco flexible, dificultan la integración de redes *on premise* y *cloud*, lo cual genera una consecuencia directa en la migración de servicios a la nube.

Las problemáticas enunciadas en el presente apartado son sólo algunas de las existentes en las redes actuales [4]. Las mismas se deben considerar como el punto de partida para justificar la generación de nuevos paradigmas y arquitecturas de *networking*. Estas deben permitir la evolución a redes que soporten el dinamismo que requieren y demandan los servicios actuales.

C. REDES DEFINIDAS POR SOFTWARE

Software Defined Networks (SDN) es un término general que se utiliza para hacer referencia a nuevos paradigmas de diseño, administración y funcionamiento de las redes. La principal propuesta de SDN se basa en la separación del plano de datos y el de control.

Ello supone un cambio disruptivo desde el punto de vista de arquitectura y funcionamiento, adaptando las redes, a la flexibilidad que se requiere a partir de la virtualización de los centros de cómputo y dotándolas de la agilidad que demanda el negocio en la provisión de nuevos servicios.

Si bien existen distintos intentos de estandarización aún disjuntos, se destaca la ONF (Open Networking Foundation) que es una organización compuesta por los mayores referentes del mercado de las telecomunicaciones [9] y cuyo objetivo es la promoción, adopción y estandarización de SDN.

La definición impulsada por dicha organización supone la separación del plano de datos y el de control, lo cual implica la existencia de un nuevo componente de red denominado controlador y de un protocolo de comunicaciones denominado OF (OpenFlow).

A continuación, se exponen los componentes que conforman una red definida por software, su funcionamiento y arquitectura.

1) Controlador

Representa el “cerebro” de la red, permite la visualización centralizada de los elementos que conforman la misma. Este comportamiento implica un cambio sustancial en lo que a arquitectura respecta.

A diferencia de las redes tradicionales cuyo funcionamiento se basa en la inteligencia distribuida y la característica monolítica de sus componentes, SDN propone la existencia de un controlador que funcione como orquestador de la red, concentrando la gestión de la misma.

De manera resumida, el controlador es el componente de red que tiene la responsabilidad de administrar los dispositivos subyacentes, indicándoles el tratamiento que se le debe dar a cada tipo de tráfico.

La existencia del controlador implica la centralización del plano de control de los equipos de red (Figura 6).

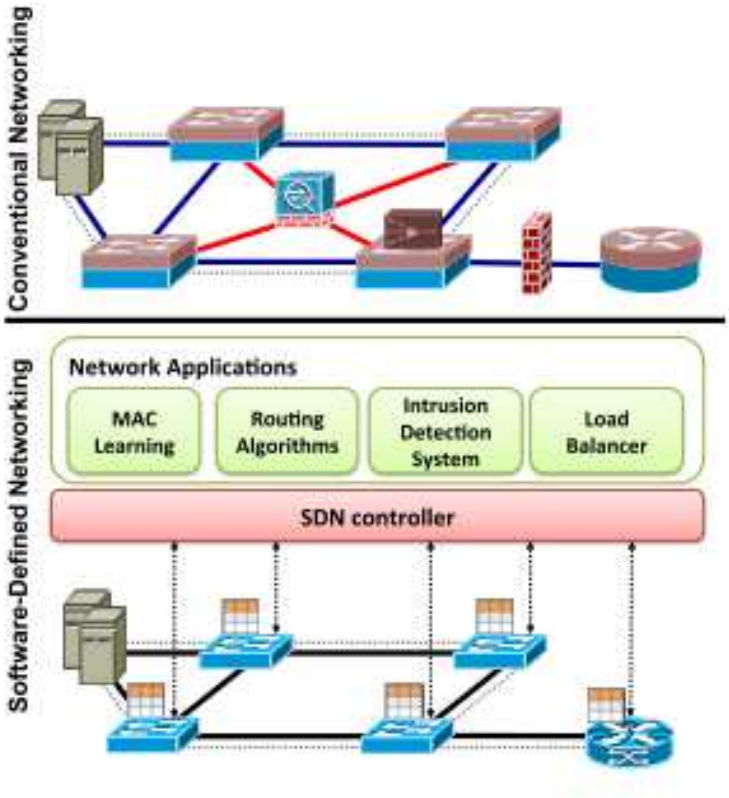


Figura 6 - Arquitectura tradicional vs. Arquitectura SDN [10]

En la actualidad existen múltiples controladores *open source* y comerciales. Algunos de ellos, se exponen en la Tabla 1.

Tabla 1 - Controladores SDN comerciales y open source [11]

Comercial	Open Source
Brocade	BEEM Controller
Ericsson	LOOM
NEC ProgrammableFlow	NOX
Sonus NaaS IQ	Ryu
Avaya SDN Fx Controller	OpenDayLight
Big Cloud Fabric	ONOS
Ciena Agility	POX
OneController	Floodlight

HP VAN Controller	
Huawei Agile Controller	
Inocybe OpenDayLight	
Nauge Networks Virtualized Service Controller	
VMware NSX Virtual Network Platform	

2) Southbound API

Representa la interfaz de comunicación entre el controlador y los dispositivos de red. Si bien existen diversos protocolos definidos para ésta, se destaca OF estandarizado por la ONF.

i. *OpenFlow*

Es un protocolo de *Networking* [5][12] que surgió como punto de partida de SDN y que hoy posibilita su implementación. El mismo especifica la comunicación entre el controlador y los switches, como así también el tratamiento que estos últimos le deben dar a los paquetes que ingresan por sus puertos.

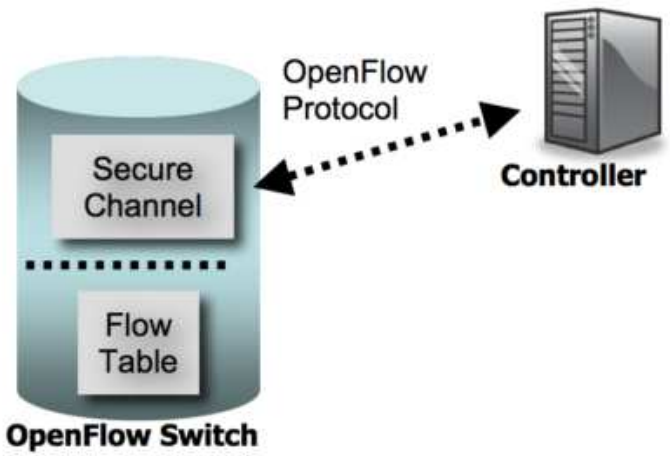


Figura 7 - Arquitectura OpenFlow [13]

La Figura 7 representa la arquitectura básica sobre la que basa su funcionamiento OF [13].

Los switches de SDN tienen al igual que los tradicionales, la funcionalidad de *forwarding* que es la que permite decidir qué hacer con cada flujo que ingresa por sus puertos. Sin embargo, a diferencia de los tradicionales que funcionan a través de MAC *learning* (proceso mediante el cual construyen su tabla de forwarding), los switches de SDN funcionan a partir de una tabla de flujo o *flow table* (Figura 9). Dicha tabla de flujo es gestionada a través del controlador.

La tabla de flujos está constituida por un conjunto o colección de entradas (flow *entries*), cada una de las cuales incluye tres campos (versión OF v1.0 [13]): *header*, *counters* y *actions*.

El campo header define un conjunto de valores contra los cuales se comparará el encabezado de los paquetes que ingresan por un puerto del switch. Los permitidos en la v1.0 del protocolo se exponen en la Figura 8.

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN pri-or-ity	IP src	IP dst	IP proto	IP ToS bits	TCP/ UDP src port	TCP/ UDP dst port

Figura 8 - Campos de las cabeceras de los paquetes usados para comparar contra las entradas de flujo en OF 1.0 [13]

Se debe destacar que una entrada de flujo no necesariamente debe contemplar todos los campos, sino que basta con que se le asigne un valor a al menos una de las variables de las permitidas de acuerdo a la Figura 8. La definición de una entrada de flujo permite la comparación respecto al encabezado de los paquetes entrantes y posibilita la clasificación de los mismos.

El campo actions indica que acción realizará el switch *ante la coincidencia del campo header con el encabezado de un paquete entrante*. En caso de que para una coincidencia no se indique ninguna acción, el switch descartará el paquete (*drop*). En caso contrario deberá indicarse una acción. El estándar OF [13] especifica las acciones requeridas y opcionales que deben ser incorporadas por un switch de SDN.

En la Figura 9 se ejemplifica la configuración de una tabla de flujos.

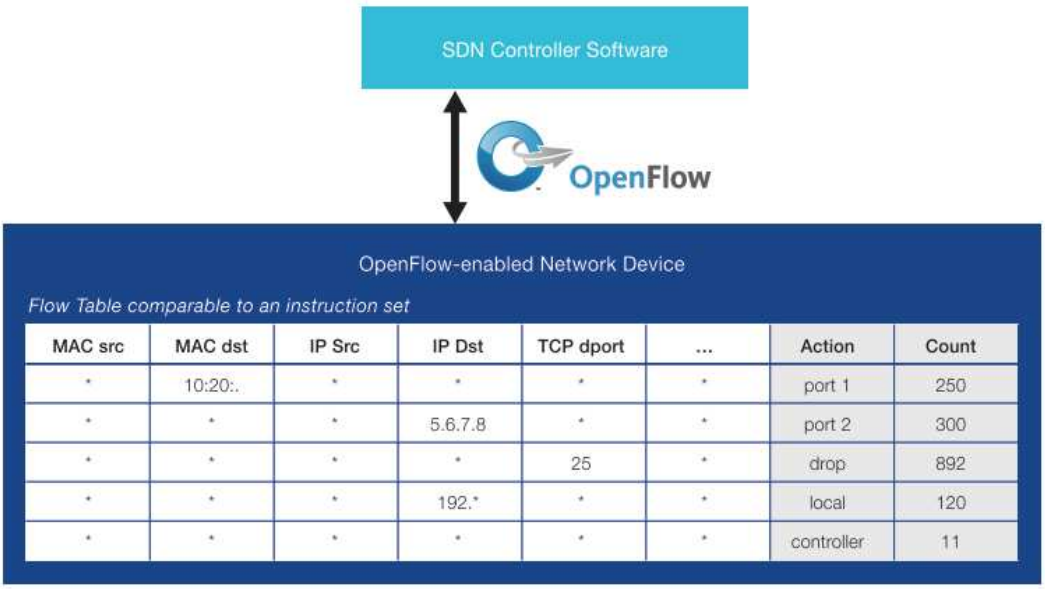


Figura 9 - Ejemplo de una tabla de flujo [4]

La tabla de flujo se recorre en forma ordenada y secuencial hasta que se produzca una coincidencia. En caso de no encontrarse ninguna, el paquete es enviado al controlador que, luego, deberá responder con un mensaje de modificación de la tabla de flujos con una nueva regla que permita el encaminamiento de dicho paquete.

En la Figura 10 se puede apreciar este proceso.

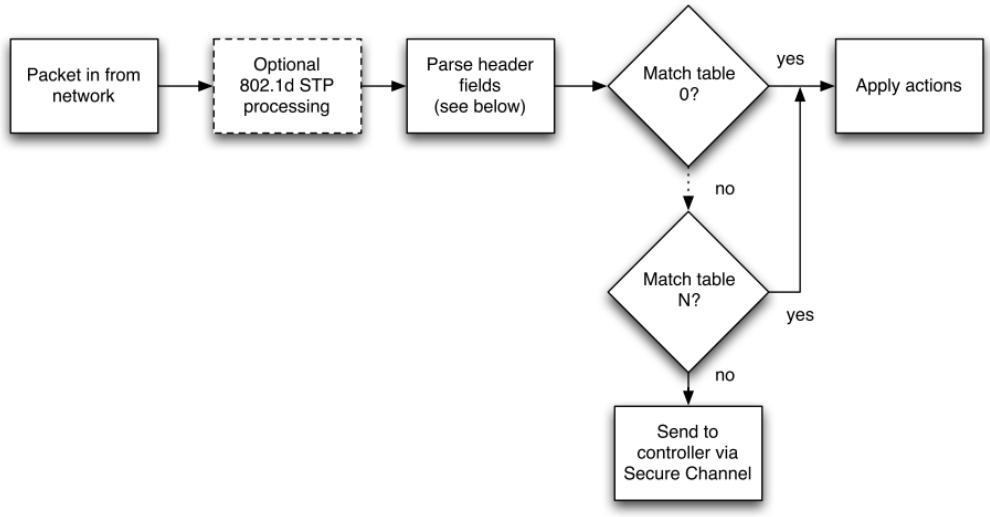


Figura 10 - Proceso de decisión de un switch OpenFlow v1.0 [13]

Actualmente OF ha evolucionado hasta la versión v1.5. Sin embargo, de acuerdo al relevamiento de equipos y controladores con soporte OF realizado, las v1.0 y 1.3 son las implementadas por la mayoría.

Adicionalmente OF define una serie de contadores que se actualizan de acuerdo a los paquetes que ingresan al switch. Los mismos se encuentran definidos en el estándar *OpenFlow Switch Specification* v1.0 [13].

- Nota: En este apartado se expone como referencia el funcionamiento de OF 1.0 puesto que es la versión compatible con las estrategias de virtualización abordadas en el presente trabajo.

3) Northbound API

La interfaz norte o *Northbound API* se utiliza para la interacción del controlador con las aplicaciones que permiten la implementación de diversos servicios a ejecutarse sobre la red. Dicha interacción resulta fundamental debido a que las aplicaciones posibilitan funcionalidades de valor agregado en SDN.

Existen aplicaciones para diversos casos de uso como ser: monitoreo, implementación de políticas (QoS), seguridad, etcétera [14]. Esto se traduce en redes programables, más flexibles y capaces de adaptarse rápidamente a las demandas de los usuarios.

Las mismas se instalan en el controlador o más comúnmente funcionan interactuando con él, como aplicativos independientes. Una premisa fundamental en el segundo caso, implica que las aplicaciones se comuniquen en forma directa con el controlador a través de interfaces estándares, abiertas y no propietarias como por ejemplo REST API [15]. La existencia de una interfaz con éstas características es la principal razón que permite la abstracción de las aplicaciones respecto de los dispositivos subyacentes y la topología a administrar, como así también del controlador. Esto facilita el diseño, desarrollo e implementación de las mismas, como así también la realización de pruebas, investigación y desarrollo en la materia.

En la Figura 11 se presenta un diagrama resumido de arquitectura, en donde se pueden apreciar los componentes que conforman una red SDN:

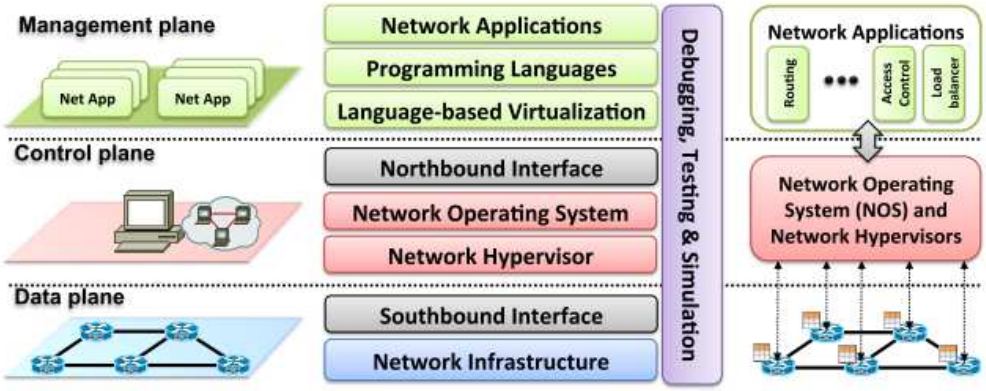


Figura 11 - Arquitectura SDN [10]

La evolución que propone SDN resulta ser similar a lo acontecido en materia de soluciones de cómputo tal como se presentó en la Figura 1. A continuación, se expone dicha analogía que además permite sintetizar las ventajas de este nuevo paradigma de *networking* (Figura 12).

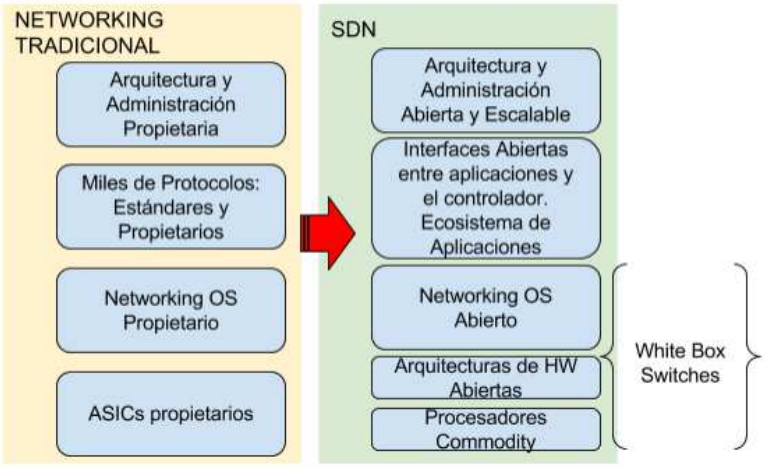


Figura 12 - Evolución de la arquitecura de networking propuesta por SDN

4) Ventajas respecto de las arquitecturas tradicionales

Se exponen de manera resumida las ventajas que ofrece SDN en relación a las arquitecturas tradicionales de *networking* [16]:

i. *Programabilidad de las redes.*

La existencia de un plano de control separado del de datos y las interfaces abiertas norte y sur, permiten la programación y configuración automatizada de las redes.

ii. *Gestión centralizada.*

La inteligencia y las políticas de funcionamiento de las redes se encuentran centralizadas en el controlador. La centralización en una única entidad facilita la configuración y administración global de la red.

iii. *Reducción de CAPEX.*

El mercado de equipamiento de *networking* hoy está concentrado en los grandes incumbentes. Las arquitecturas son cerradas y propietarias, lo que genera una consecuente dependencia y elevados costos. SDN propone una arquitectura con interfaces abiertas, lo que posibilita la apertura del mercado a nueva competencia posibilitando la reducción de costos de adquisición de tecnología de networking.

iv. *Reducción de OPEX*

La gestión centralizada posibilita la reducción de costos al simplificar los procesos de administración, configuración y despliegue de nuevas funcionalidades en las redes de datos.

v. *Agilidad y flexibilidad en los nuevos despliegues*

SDN facilita a través de sus interfaces abiertas el despliegue de nueva infraestructura, servicios y aplicaciones, lo cual permite mayor agilidad, escalabilidad y flexibilidad en la adaptación de las redes a las demandas cambiantes del mercado.

vi. *Permite la innovación*

La existencia de interfaces abiertas posibilita la apertura de las tecnologías de *networking* en un camino hacia la innovación, posibilitando que organizaciones educativas y nuevos emergentes puedan aportar valor agregado en este proceso de evolución.

2.VIRTUALIZACIÓN

A. ANTECEDENTES

La evolución de las tecnologías de cómputo, *storage* y *networking*, permiten vislumbrar una tendencia irreversible hacia el aprovisionamiento más dinámico y la configuración automatizada de la infraestructura de *datacenter*. Esto se pone de manifiesto principalmente en las soluciones actuales de virtualización de servidores [17][18][19][20], dominio en el cual existe mayor madurez.

El grado de desarrollo de dichas soluciones, ha llevado a que hoy en día los centros de cómputo estén compuestos de manera predominante por un gran número de servidores virtuales, que superan ampliamente a los físicos. Ésta situación se ha ido consolidando a lo largo del tiempo, tal como se muestra en la Figura 13.

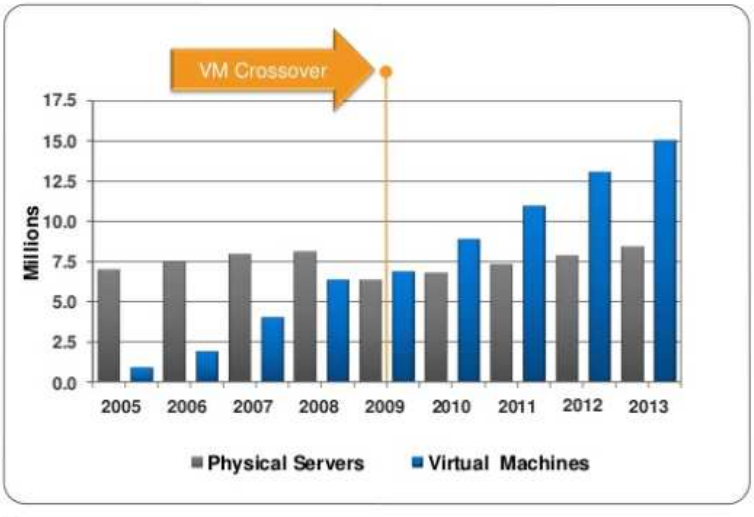


Figura 13 - Physical Servers vs. Virtual Machines - Fuente IDC/vmware

El concepto de virtualización, implica la utilización más eficiente de la infraestructura física a través de la generación de una o más abstracciones virtuales (basadas en software) que permiten su uso compartido. Cada una de las instancias virtuales, desconoce dicha condición, presentándose como si fuera la única de cara al usuario. El diccionario Oxford [21] define la palabra "*virtual*", en el contexto de computación, de la siguiente manera: "*Not physically existing as such but made by software to appear to do so: virtual images*"

La virtualización, agiliza los procesos de aprovisionamiento, permitiendo afrontar la dinámica que imponen las actuales cargas de trabajo. Esto se manifiesta en la posibilidad de prestar servicios innovadores reduciendo el "time to market", entre otros [22].

Actualmente existen varias siglas que representan este paradigma, SDx, IaaS, SDE [23], pero que en definitiva denotan el mismo concepto a partir del cual se intenta abstraer a los servicios, de la infraestructura subyacente. Una definición más rigurosa, permite afirmar que un "ambiente definido por software" es aquel en el cual las cargas de trabajo son manejadas en forma independiente de la infraestructura, permitiendo la asignación de recursos de manera dinámica, en función del grado de utilización de los mismos y con una mínima intervención por parte del administrador.

En este sentido, los avances recientes en lo que se refiere a soluciones de virtualización de cómputo y de storage nos ha permitido acercarnos a este paradigma. El último eslabón sobre el cual existen aún avances prematuros en pos de lograr infraestructuras totalmente virtualizadas, se presenta en el campo del *networking*.

En este contexto, SDN aparece como un paradigma novedoso y prometedor en la evolución de las arquitecturas de redes tradicionales, con la posibilidad de generar soluciones que permitan la completa virtualización de los centros de cómputo o SDDC (Software-Defined Data Center).

B. ORÍGENES DE LA VIRTUALIZACIÓN

1) Virtualización de Storage

El concepto de virtualización de *storage* se refiere a la separación entre el espacio físico disponible en disco respecto de la asignación lógica de dicho espacio. De esta manera, la capacidad disponible en múltiples discos se puede combinar en una única LUN

(Logical Unit Number). Para ello, debe existir un software de virtualización que mantenga el registro del mapeo lógico vs. físico realizado, utilizando meta-data.

La virtualización de *storage* aparece como consecuencia de la utilización ineficiente de los recursos de almacenamiento. El agrupamiento lógico de múltiples unidades físicas facilita la administración optimizando la utilización del espacio disponible [24] como así también la asignación flexible de espacio en disco a los usuarios. Por otra parte, posibilita procesos de migración no disruptivos. Ello permite la reducción de tiempos asociados a este tipo de proceso, reduciendo al mínimo los requerimientos de *downtime* de las aplicaciones y por consiguiente minimizando los costos de migración [25].

De esta manera los usuarios finales tendrán presentadas unidades lógicas sobre las cuales se ejecutarán operaciones de I/O (Input/Output). Estas deben ser traducidas en requerimientos de I/O reales sobre los discos físicos, para lo cual el software de virtualización deberá realizar el mapeo correspondiente utilizando la información en la meta-data. Este proceso se denomina "*I/O redirection*".

En resumen, la virtualización de storage genera los siguientes beneficios.

i. *Facilita el aprovisionamiento de espacio de almacenamiento*
ii. *Permite procesos de migración no disruptivos*
iii. *Facilita la administración*

2) Virtualización de Servidores

La virtualización de servidores es el mecanismo por el cual los recursos de hardware son particionados en múltiples instancias lógicas o virtuales. Cada uno de los servidores virtuales o VM (Virtual Machine) tendrá asignado una parte de los recursos físicos, sistema operativo propio y aplicaciones específicas instaladas en él. La característica de aislación propia de este tipo de tecnologías permite que la virtualización sea totalmente transparente al usuario.

La virtualización de servidores posibilita la consolidación [17]. Este concepto toma trascendencia en los centros de cómputo actuales, ya que posibilita la mayor y mejor utilización de los recursos de hardware e infraestructura. Adicionalmente facilita la administración; permite la migración de una VM hacia otro servidor físico sin disrupciones en el servicio, posibilita la automatización en la asignación de nuevos

recursos y minimiza los cortes de servicio ante falla debido a la característica de resiliencia propia de éste tipo de tecnologías.

La virtualización requiere en general de la existencia de un hipervisor o VMM (Virtual Machine Monitor) que es una capa intermedia entre cada máquina virtual y el hardware subyacente [26]. La responsabilidad de esta capa es la de administrar los requerimientos de I/O de cada una de las VMs. De acuerdo a la tecnología de virtualización utilizada se requiere o no de un sistema operativo de base en el equipo físico (Figura 14). A modo de ejemplo KVM [20] requiere de la instalación del OS mientras que vmware ESX [27] y Microsoft Hyper-V [28] no lo requieren.

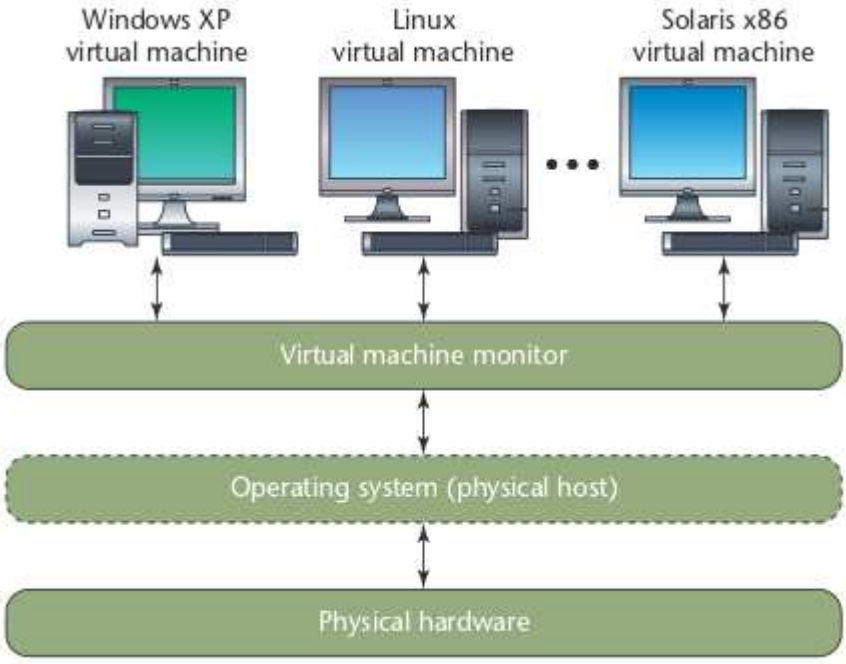


Figura 14- Arquitectura de una solución de virtualización de servidores [26]

La transparencia al usuario no es aplicable a todas las tecnologías de virtualización. Éste es el caso de XEN [19] que utiliza el concepto de paravirtualización. La paravirtualización requiere de la comunicación entre el OS (Operating System) de la VM y el VMM. El VMM puede delegar tareas críticas (con mucho requerimiento de I/O) para que sean ejecutadas directamente por el dominio físico y no por el virtual. Tal como se expresó, este proceso no es transparente al usuario puesto que se requieren de modificaciones de OS, siendo su principal ventaja que genera menor *overhead* [29].

La virtualización basada en contenedores o *containers* es una técnica de virtualización a nivel de sistema operativo [30]. Los contenedores permiten que múltiples instancias del

mismo OS se ejecuten en el espacio de usuario. En este caso, es el kernel del OS subyacente, el que tiene la responsabilidad de crear y administrar los recursos para la ejecución de las múltiples instancias en forma simultánea. Esta alternativa es la que genera menos overhead si se la compara con las otras estrategias abordadas.

Una aplicación de virtualización basada en contenedores es Docker [31]. Esta solución utiliza *containers* para virtualizar una instancia de OS con aplicaciones pre instaladas y sus dependencias, permitiendo "empaquetarlo y desplegarlo" en cualquier otro OS de manera rápida. A su vez, la solución cuenta con un sitio llamado "Docker Hub"[32] el cual sirve de repositorio para el almacenamiento y descarga de estos contenedores, permitiendo el despliegue de nuevos entornos virtuales en cuestión de minutos y gran flexibilidad ante requerimientos de portabilidad de los mismos.

C. PRIMEROS PASOS HACIA LA VIRTUALIZACIÓN DE LAS REDES

En forma general la virtualización es una tecnología de software que posibilita el desacoplamiento de la infraestructura física a través de la creación de instancias lógicas de la misma, que en el contexto de *networking* se traducen en mecanismos de *forwarding* y direccionamiento propios [33]. Ello se logra a través de la abstracción de los servicios respecto de los recursos físicos subyacentes, otorgando así flexibilidad.

Este concepto se presenta en la Figura 15.

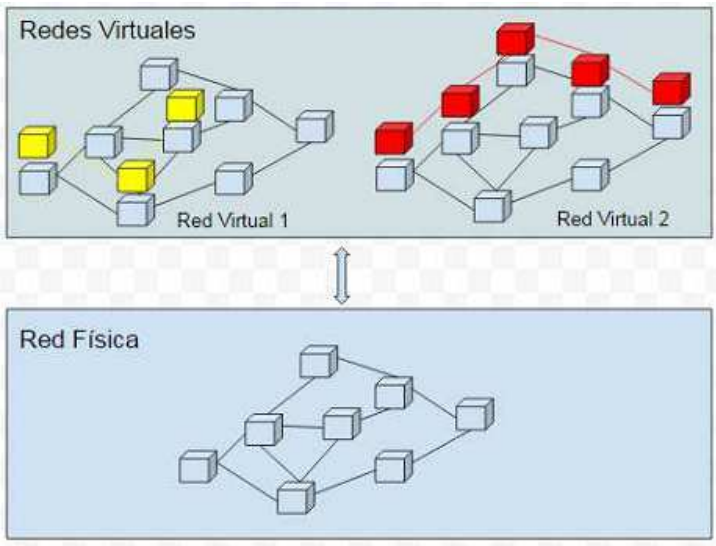


Figura 15 - Virtualizacion de una red

En particular, la virtualización de una red al igual que en el caso de servidores permite que múltiples usuarios o *tenants* utilicen recursos de la misma infraestructura, sin capacidad para identificar el uso compartido.

La virtualización de las redes ha sido un objetivo de las comunidades de investigación y desarrollo, puesto que posibilita la realización de pruebas sobre entornos productivos en forma directa, sin afectar ni interferir en el tráfico y su funcionamiento normal [33]. Sin embargo, el interés por la virtualización de redes no sólo deviene de los académicos, sino que los proveedores de servicios de telecomunicaciones tienen grandes expectativas en la materia. En este caso, el objetivo es poder agilizar la provisión de servicios, implementar aislación lógica entre usuarios, entre otros.

Es así que, en el contexto de redes tradicionales, se han desarrollado múltiples técnicas que permiten explotar con algunas limitaciones dichos atributos de la virtualización, algunas de las cuales se introducirán de manera breve en el siguiente apartado.

1) Virtualización en redes tradicionales

i. *VRF Lite*

Virtual Routing and forwarding (VRF) es una tecnología que permite la virtualización de los dispositivos de red de capa 3. La consecuencia directa de ello es que un único dispositivo físico funciona como múltiples dispositivos lógicos independientes tal como se aprecia en la Figura 16. Específicamente permite la creación de múltiples tablas de ruteo en un único *router*.

Sin embargo, no se puede considerar a una VRF como un *router* virtual en su totalidad [34], puesto que dichas tablas comparten el mismo HW y en consecuencia sus recursos (procesamiento, memoria, puertos físicos, etc).

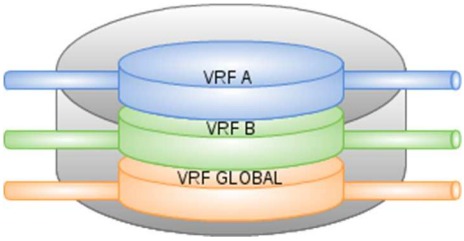


Figura 16 - Virtualización de dispositivos de capa 3

La configuración de diversas VRFs en un *router* posibilita la utilización de una misma dirección IP (Internet Protocol) en dos interfaces físicas de un mismo equipo. Es decir que, se podría utilizar la dirección IP 192.168.0.1 en la interfaz correspondiente a la VRF A y también en la correspondiente a la VRF B.

Para posibilitar la creación de una VPN (Virtual Private Network), cada una de las instancias o VRFs creadas deben ser interconectadas entre sí. Esto se logra a través de la conexión física entre ellas, lo cual resulta muy ineficiente y costoso. A partir de ello, resulta necesaria la virtualización de dichas conexiones, para lo cual se suele utilizar IEEE 802.1Q (Figura 17) en caso de que no existan saltos intermedios. De requerirse la virtualización de los caminos físicos entre dos dispositivos interconectados a través de múltiples saltos, se puede optar por un protocolo de tunelización como por ejemplo Generic Routing Encapsulation (GRE).

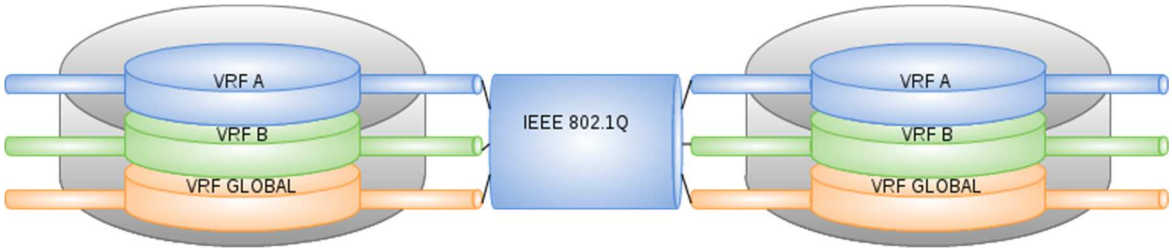


Figura 17 – Virtualización de interconexiones físicas a través de dot1Q

La mayor limitación de las VPNs creadas a partir de la utilización de VRFs es su capacidad para escalar, debido a cualquier modificación en la configuración de la red virtual, implica la reconfiguración manual de cada uno de los de dispositivos de red que la conforman. Los mismos, se debe reconfigurar en los siguientes aspectos:

Virtualización de los dispositivos de red: requiere la configuración manual de VRFs en los dispositivos de capa 3 o de VLANs asociadas los dispositivos de capa 2. Las VRFs configuradas se deben replicar en forma manual en cada uno de los *routers* de la VPN.

Virtualización de la interconexión entre los dispositivos: requiere la configuración del protocolo IEEE 802.1Q[1] en cada uno de los *uplinks* de los dispositivos de red que

1 IEEE 802.1Q o dot1Q, permita a múltiples redes compartir de forma transparente el mismo medio físico (virtualización del medio físico) separando el tráfico correspondiente a cada una de las redes (Trunking)

conforman la VPN. Esta configuración permite la generación de *trunks* de capa 2 que posibilita la segmentación lógica del tráfico correspondiente a cada VRF. Para ello cada instancia virtual de capa 3 debe ser mapeada con una determinada VLAN, lo que permite la generación de VPNs extremo a extremo (Figura 18).

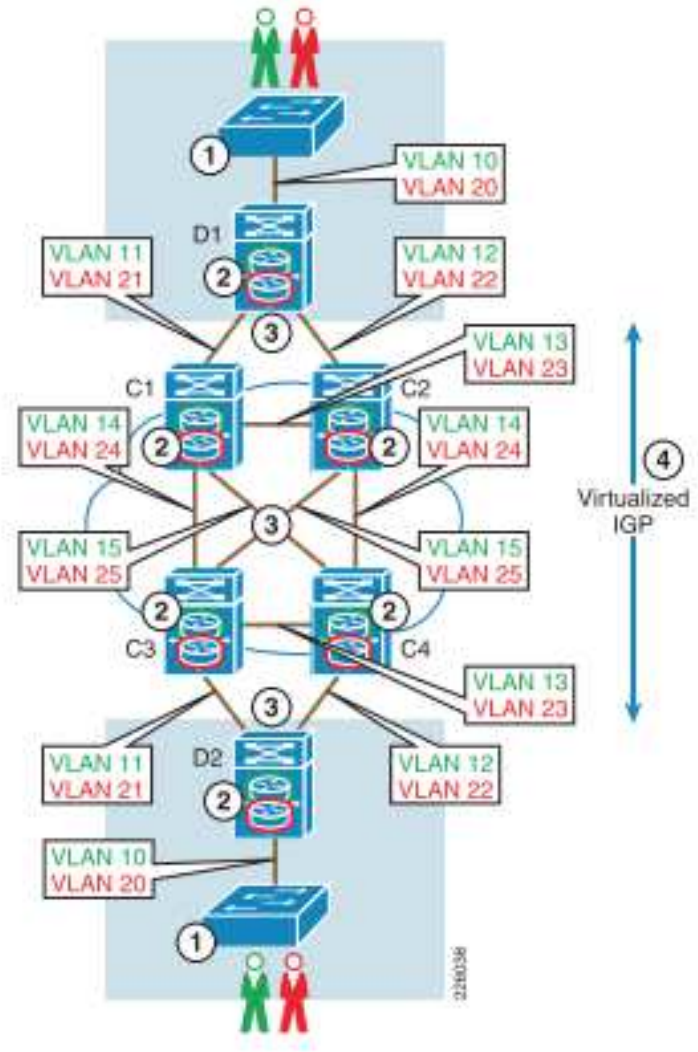


Figura 18 - Configuración VRF Lite extremo a extremo de la red [34]

En este apartado se ejemplifica la virtualización de redes tradicionales a través de VRFs, sin embargo, existen otras como ser:

- WDM, tecnología que permite virtualizar las conexiones físicas ópticas, a través de la utilización de múltiples longitudes de onda.
- VLAN, permite la segmentación de los dominios de broadcast lo cual permite la creación de múltiples redes lógicas de capa 2.

Si bien las técnicas de virtualización expuestas para redes tradicionales son de basta utilización en la actualidad, las mismas se ven condicionadas por las problemáticas presentadas en 2)3) Problemas de las redes de datos tradicionales.

En contraposición, tal como se presenta en 2.VIRTUALIZACIÓN apartado a.Antecedentes, SDN aparece como un paradigma novedoso y prometedor en la evolución de las arquitecturas de redes tradicionales, con la posibilidad de generar soluciones más flexibles para la virtualización de las redes.

D. SDN EN LA VIRTUALIZACIÓN DE LAS REDES

SDN propone a través de su arquitectura la separación del plano de datos y el de control, centralizando este último. Tal como se expuso en 1.SDN apartado c.Redes definidas por Software, dicha centralización se da a través de un nuevo componente denominado controlador, el cual posibilita la visión y gestión sobre la totalidad de la red. Ello permite, al igual que en el caso de las soluciones de cómputo, la existencia de la capa de virtualización o "*network hypervisor*" [35], que es en definitiva la que habilita la abstracción de la capa física subyacente. Es así, que la posibilidad de virtualizar la red es uno de los casos de uso más prometedores de SDN.

Tal como se expresó en b. Orígenes de la virtualización, esta tecnología admite que múltiples usuarios puedan disponer de recursos de procesamiento o storage bajo demanda, en forma dinámica y ágil, simplificando además los procesos de administración y migración. Este escenario no es posible bajo el paradigma aún vigente de Networking.

Las arquitecturas tradicionales suponen configuraciones manuales e individuales para cada equipo dentro de la red. Tal como se expresó en 1.SDN apartado b.Redes de datos tradicionales, cada elemento toma decisiones de manera autónoma e independiente, basándose en protocolos de *forwarding* y ruteo, que permiten la correcta transmisión de datos extremo a extremo.

Se pueden considerar como conceptos "primitivos" en lo que a virtualización de las redes se refiere a: la segmentación de capa 2 a través de VLANs, la generación de circuitos virtuales a través de MPLS (Multiprotocol Label Switching), la implementación

de “routers” virtuales a través de VRFs, VPNs, entre otros, algunos de los cuales fueron tratados en el apartado b. Orígenes de la virtualización.

Se debe destacar nuevamente que, la implementación de estos protocolos o soluciones implica el diseño en detalle de la red. Luego se debe configurar en forma individual a cada componente para que la misma funcione de acuerdo a las características de diseño. Como consecuencia, los tiempos de aprovisionamiento de infraestructura de networking son prolongados, siendo no comparables con los relativos a infraestructura virtualizada de cómputo y storage [36].

Los futuros mecanismos de virtualización deberán permitir flexibilidad en dos sentidos: topología y esquemas de direccionamiento. En contraposición, hoy son en esencia estáticas y con esquemas de direccionamiento fijos (IP, MAC, VLAN ID, etc).

SDN ha abierto las puertas a un nuevo paradigma de networking, generando un impulso en los ambientes de investigación y desarrollo, en pos de obtener nuevas funcionalidades y protocolos que simplifiquen, agilicen y les otorguen flexibilidad a las redes. Ello se ve manifestado a partir de la definición de estrategias que marcan una tendencia clara hacia la virtualización de las mismas [37] [38] [39] [40].

Adicionalmente la utilización de OF como protocolo estándar entre el controlador y la red física posibilita la abstracción de los dispositivos subyacentes facilitando así, la generación de instancias virtuales de la red. Siguiendo el modelo de virtualización de cómputo, si se incorpora a la arquitectura tradicional de SDN una capa de virtualización, es posible la creación de redes virtuales (Figura 19).

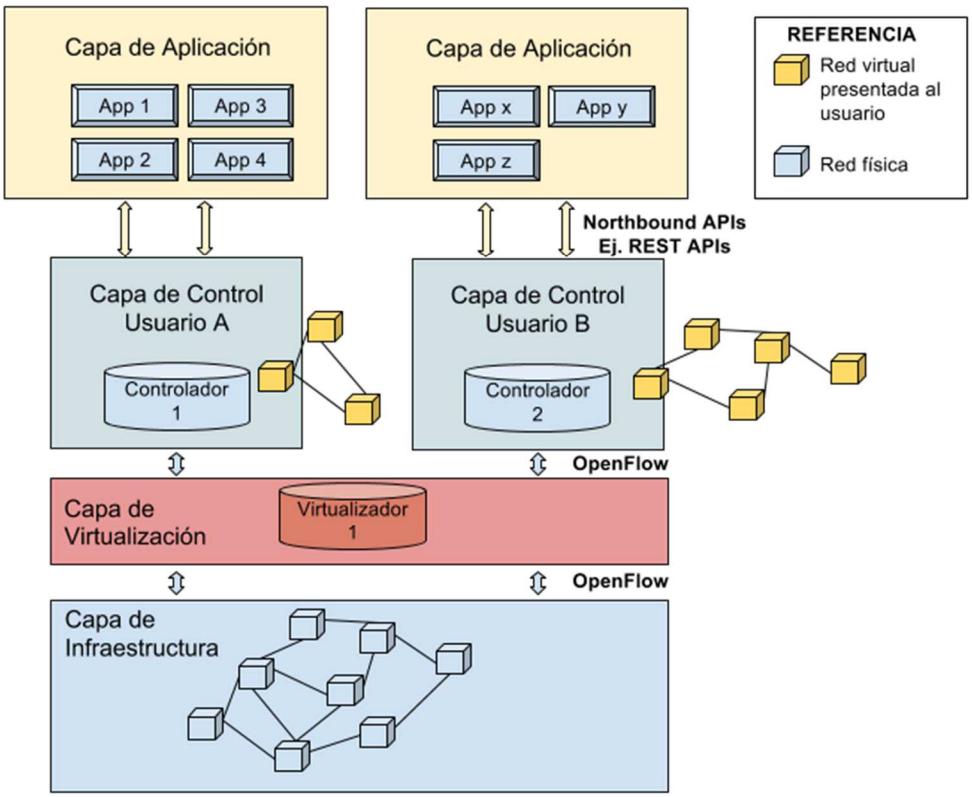


Figura 19 - Arquitecura de red SDN virtualizada

En un contexto de redes definidas por software, la virtualización de una red permite compartir el recurso físico entre grupos de usuarios o *tenants*, sin que esto se evidencie a nivel de usuario final. Ello implica que cada instancia de red cuente con su propio controlador, así como cada VM con su propio sistema operativo, lo cual posibilita el funcionamiento en paralelo de las mismas.

III.DESARROLLO

1.DEFINICIÓN DEL PROBLEMA

A. DEFINICIÓN DEL PROBLEMA A ABORDAR

El presente trabajo aborda la *intersección entre las tecnologías SDN y virtualización de redes.* Se definen atributos requeridos para una solución de virtualización que permita el desarrollo de un *framework* que sirva para caracterizarlas. Estos atributos se utilizarán para la comparación de algunas de las técnicas de virtualización existentes para SDN.

Tal como se detalló en el capítulo II.MARCO TEÓRICO, existe un amplio interés por parte de la industria de las telecomunicaciones y las comunidades de investigación y desarrollo en las ventajas que brindan las soluciones de virtualización, en particular a partir de la aparición de este nuevo paradigma. En este contexto, se propone identificar de qué manera la implementación de SDN posibilita la virtualización de las redes.

Sin embargo, resulta difícil determinar aquellas soluciones que tienen posibilidades de ser desplegadas en el futuro en redes productivas de gran escala, ya que esto depende de muchos factores [41]. Para ello, se analizarán diferentes definiciones relativas a la virtualización de las redes; y se buscará consolidar una definición objetiva que permita determinar unívocamente aquellas características y funcionalidades, que necesariamente deben ser incluidas en una solución integral. Así, se definirán atributos relativos a las soluciones de virtualización y que resulten de interés para el Marco de trabajo.

En este sentido se estudiarán y caracterizarán propuestas existentes en el contexto de SDN, centrando el estudio en las siguientes:

a. FlowVisor [33]
b. OpenVirtex [42][43]

La selección se basó en que ambas soluciones utilizan la manipulación de flujos OF para lograr la virtualización, lo cual se destaca como una característica innovadora. Además, tienen un grado de desarrollo tal que posibilita su implementación y la realización de simulaciones que permitan el estudio propuesto.

Se las caracterizará de acuerdo a los atributos definidos, analizando bibliografía existente, publicaciones científicas y mediante la realización de simulaciones que

permitan la descripción objetiva de las mismas, su comparación y determinación de ventajas y desventajas.

El presente trabajo aportará valor para clasificar y comprender el grado de madurez de las soluciones de virtualización basadas en SDN en general y de las propuestas abordadas en particular. A partir de dicho análisis, se buscarán identificar además las ventajas comparativas respecto de las soluciones de virtualización en redes tradicionales.

La definición de atributos permitirá el desarrollo de un *framework*, que será de utilidad para evaluar de manera general otras propuestas de virtualización en SDN actuales o futuras. Dicho *framework* y la caracterización de las técnicas de virtualización abordadas serán *el aporte al estado del arte* del presente trabajo.

Adicionalmente, tal como se expone en "Software-Defined Networking: A Comprehensive Survey"[10], la temática representa un campo de actualidad, donde aún existen muchos desafíos y se requieren esfuerzos de investigación.

B. MARCO DE TRABAJO

Si bien el concepto de virtualización de las redes no está acotado a una parte específica de las redes de datos, existe un interés particular en lo que respecta a los centros de cómputo. Así lo demuestra una encuesta realizada por SDxCentral [16] (Año 2015) en la cual el 46% de los consultados implementó o realizó pruebas de concepto sobre alguna de las soluciones de virtualización, de los cuales el 48% lo realizó en estas instalaciones en particular [44].

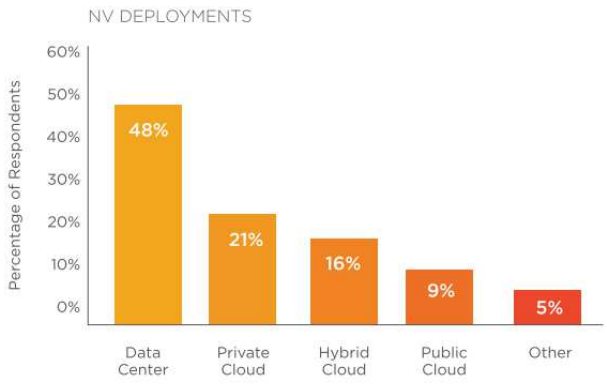


Figura 20 - ¿Dónde se están desplegando soluciones de virtualización de redes?

[44]

Es por ello que el presente trabajo se centra en el estudio de soluciones de virtualización en SDN con vistas a las ventajas en las redes de datos de los centros de cómputo. Se le da este enfoque debido que es el contexto en donde se identifica de manera clara la necesidad de evolución de las redes. Esta decisión fue en parte expuesta y justificada en el capítulo II, apartados 1)Virtualización de Storage y 2)Virtualización de Servidores.

Se focalizará en los *datacenter* para la definición de atributos puesto que parece ser el caso de uso de mayor impacto en la industria. Sin embargo, la definición de este marco no implica que los resultados obtenidos a partir del presente estén acotados a este único escenario.

En resumen, a través del presente trabajo se buscará aportar valor en relación a los siguientes puntos:

a. Realizar un relevamiento del estado del arte de propuestas de virtualización en redes SDN.
b. Definir atributos que son necesarios y deseables para una solución de virtualización.
c. Realizar una caracterización y comparación de las propuestas de virtualización seleccionadas.
d. Aportar un *framework* que sirva para evaluar de manera general estrategias de virtualización en redes SDN

2. NUEVAS ESTRATEGIAS DE VIRTUALIZACIÓN

En el contexto de redes SDN el controlador es el responsable de las decisiones de *forwarding* y *ruteo* a partir de la administración y carga de la tabla de flujos en cada uno de los dispositivos de red bajo su control. Dicho proceso se realiza a través de la utilización del protocolo OF, tal como se presentó en el capítulo II.MARCO TEÓRICO apartado c. Redes definidas por Software.

Basándose en este concepto y en la arquitectura de SDN, si se coloca una capa de virtualización ubicada lógicamente entre el controlador y los dispositivos de red cuyas interfaces utilicen OF, será posible segmentar la red. Esta funcionalidad se implementa a través de un hipervisor con capacidades de manipulación de dicho tráfico (Figura 21). Ello permite que los controladores asociados a una instancia virtual sólo puedan administrar los switches y los flujos correspondientes bajo su dominio. Para implementar

esta lógica, la capa de virtualización mantiene información actualizada de dicho mapeo, alterando el tráfico OF según corresponda. Lo expuesto resume el principio de funcionamiento de las soluciones de virtualización a abordar, lo que permite configurar una arquitectura análoga a la presentada en la Figura 19.

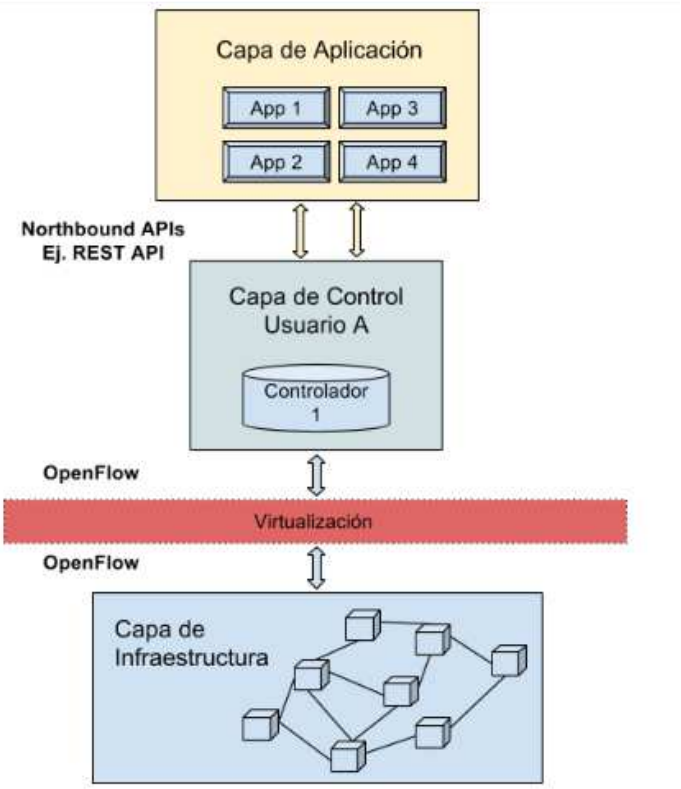


Figura 21 – Capa de virtualización o hipervisor con interfaces OF

A continuación, se detallan características específicas asociadas a FV y OVX. El objetivo de dicha exposición no es suplir la documentación técnica disponible para cada caso [33][43][42][45][46], sino servir de contexto para hacer más comprensible la problemática abordada a través del presente.

A. FLOWVISOR

FV es una solución que posibilita la abstracción de una red física en múltiples redes lógicas funcionando como capa de virtualización. Está basada en el paradigma de redes definidas por software (SDN), en el cual la funcionalidad de control se centraliza y se separa de la de *forwarding*.

Teniendo en cuenta esta premisa de diseño, FV se sitúa lógicamente entre la red física y los controladores funcionando como un hipervisor centralizado, tal como se presenta en la Figura 22. Esto la diferencia de otras soluciones de virtualización en SDN en las cuales ésta funcionalidad se implementa de manera distribuida [39][38].

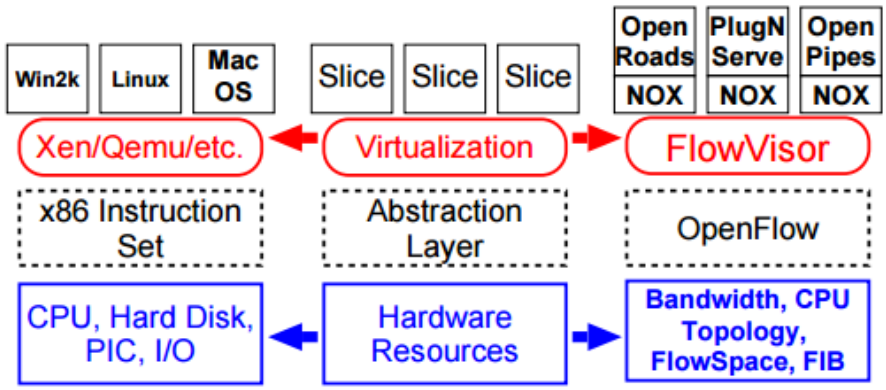


Figura 22 - Arquitectura de Flowvisor y su comparación con soluciones de virtualización de cómputo [33]

FV [33][41][47][45] surge a partir de la necesidad de utilizar una misma red física de manera compartida para producción y experimentación, lo cual requiere de un mecanismo de virtualización. Éste es uno de los principales objetivos en ambientes académicos, puesto que posibilita la realización de ensayos en redes reales, lo cual facilita los procesos de innovación e investigación en la materia.

i. *Arquitectura y Funcionamiento*

FV es una solución de virtualización basada en software que posibilita la creación de diversas redes virtuales sobre una misma red física. Dicha funcionalidad se implementa a través de un hipervisor que lógicamente se posiciona entre los dispositivos de red y los controladores. Cada una de las redes generadas estará virtualmente aislada respecto de las otras, lo que posibilita esquemas de *forwarding* y *direccionamiento propio* para cada una. A este proceso de segmentación de la red se lo denomina *slicing*. En FV, este concepto está asociado a la definición de un conjunto de flujos que se encuentran bajo control de una determinada instancia virtual de la red.

FV utiliza OpenFlow [13][5] como abstracción de los dispositivos físicos. Si se realiza una analogía con las soluciones de virtualización de cómputo, OF resulta el equivalente al set de instrucciones x86 que permite administrar y asignar de manera transparente los recursos de hardware a cada VM, tal como se presenta en la Figura 22. Así como cada máquina virtual posee su propio sistema operativo, FV permite que cada instancia de la red tenga su propio controlador. Cada uno de ellos tendrá la visión de los componentes físicos asociados a la red virtual bajo su dominio y la administración de los flujos correspondientes de acuerdo a la definición de cada *slice*.

La solución de virtualización funciona como un proxy transparente interceptando y manipulando el tráfico OF entre los controladores y los switches. Dicho proceso se realiza a partir de las políticas definidas para cada red, para lo cual se utiliza su API de administración. Tal como se aprecia en la Figura 23, cuando FV intercepta un mensaje OF desde el contralor (1) determina a que instancia de la red corresponde (2) y lo reescribe de manera transparente de acuerdo a su política (3) permitiendo que el controlador que originó dicho mensaje, solo controle los flujos correspondientes al slice que le corresponde. A su vez, y en el camino inverso, los mensajes desde los switches (4) son enviados sólo al controlador correspondiente.

De acuerdo a las políticas implementada FV puede dejar pasar el mensaje sin cambios, transcribirlo o enviar un mensaje de error OF al emisor del mismo.

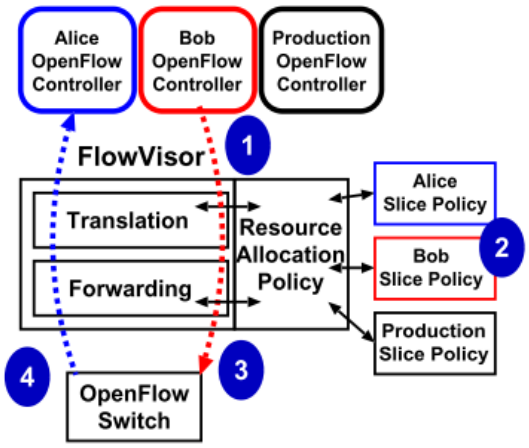


Figura 23 - Funcionamiento de Flowvisor [33]

Para los dispositivos de red FV se presenta como un controlador; mientras que para los controladores como un dispositivo de red. Esto se debe a que utiliza en ambas interfaces el protocolo OF.

Formalmente, FV define el concepto de *flowspace* [33]. Un *flowspace* es un subespacio del espacio de campos de cabecera definidos en OF (Figura 8) [13]. FV permite la generación de instancias de red virtuales a partir de la asignación de un determinado *flowspace* a cada una. Es así que, de acuerdo a la cabecera de un paquete, podrá identificar que *flowspace* lo contiene y en consecuencia a que slice pertenece.

Se debe destacar, que un paquete puede corresponder a más de una instancia. Este puede ser el caso de que se requiera una red de monitoreo, para lo cual una determinada red virtual podrá visualizar la totalidad del tráfico de la red.

B. OPENVIRTEX

OpenVirtex es una plataforma de virtualización de redes SDN. Se caracteriza por permitir generar instancias virtuales de una red, especificando para cada usuario o *tenant* su *propia topología* y *esquema de direccionamiento*, además de su propio NOS (Network Operating System) o controlador.

OVX (OpenVirtex) cumple la funcionalidad de hipervisor funcionando como proxy transparente, es decir, interceptando el tráfico OF entre los controladores y los switches. En este contexto, OVX expone redes OF a los NOS los cuales tienen la capacidad de controlar únicamente los recursos de red que le fueron asignados. Es así que los usuarios desconocen que están utilizando una red con recursos compartidos.

Una característica destacada es que cada red virtual instanciada puede utilizar la totalidad de los espacios de flujo posibles (Figura 8) de acuerdo a la especificación de OF 1.0 [13]. FV se diferencia en este punto ya que, por diseño, virtualiza a una red segmentando dicho espacio de flujos y asignando una porción del mismo a cada red virtual [43] (slicing).

i. *Arquitectura y Funcionamiento*

OVX reescribe los mensajes OF entre los controladores y los equipos de red asegurando así que el comportamiento y topología de las redes virtuales presentadas a los usuarios, sea el correspondiente a las condiciones de diseño y configuración de cada una de ellas. Esto se presenta en la Figura 24, en donde se puede apreciar, que OVX se encuentra lógicamente entre los controladores asociados a cada tenant y la red física.

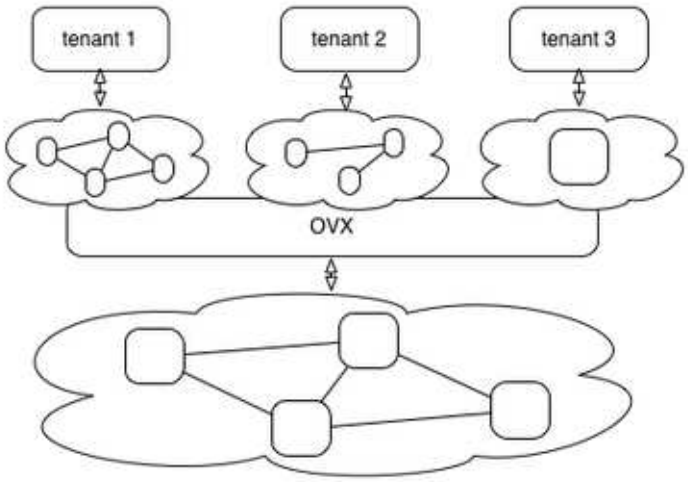


Figura 24 – Lógica de funcionamiento de OVX [46]

OVX almacena una representación de la topología física la cual obtiene a través de un mecanismo basado en LLDP (Link Layer Discovery Protocol) [46]. Asimismo, mantiene la configuración de las topologías de cada una de las redes virtuales implementadas a través de la API de administración [48].

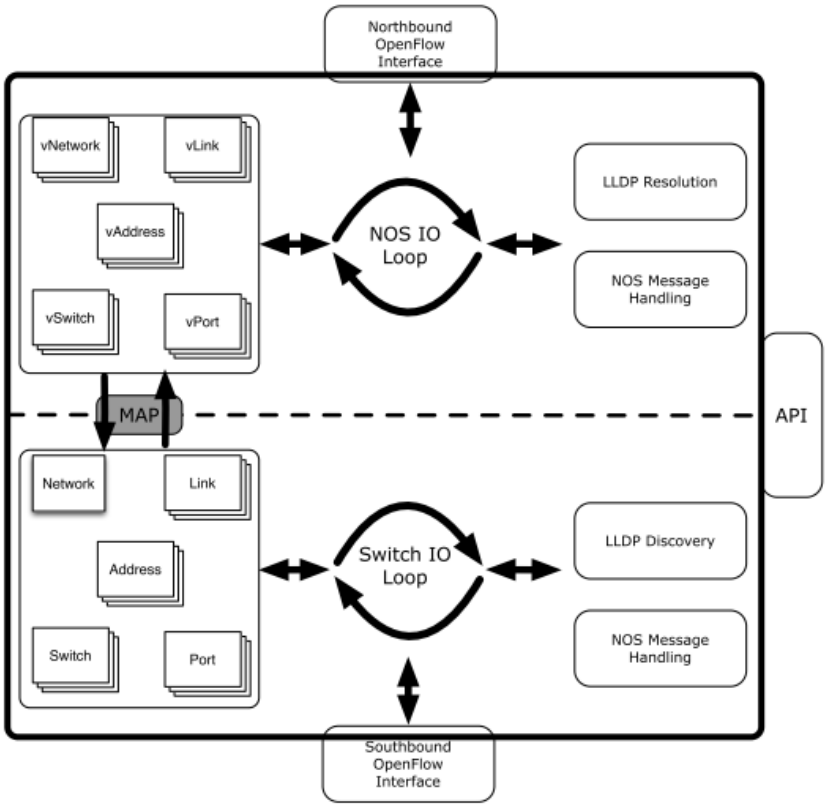


Figura 25 – Arquitectura interna de OVX [43]

Las representaciones que mantiene OVX de la red, tanto física como virtual, son colecciones de objetos que permiten instanciar redes, switches, puertos, links y direcciones. La funcionalidad *map* (Figura 25) permite el mapeo n a 1 de los objetos que representan a la red virtual y física [46] respectivamente. Todos los elementos virtuales son mapeados con al menos un elemento físico. De esta manera, OVX presenta a cada usuario una red SDN totalmente virtualizada (vSDN), permitiendo la utilización de la totalidad del espacio de flujo y otorgando la flexibilidad de generar múltiples redes virtuales con diversas topologías, sobre una misma red física. Para lograr esto, OVX resuelve mensajes LLDP provenientes de los controladores, lo que permite la exposición de la topología de cada red virtual.

El proceso de virtualización y desvirtualización presentado en la Figura 26, involucra todas las acciones para asegurar la correlación del mundo virtual y físico. Esto se manifiesta a nivel OF de la siguiente manera [46]:

- Modificación de las direcciones de red de origen y destino.
- Asociación switch físico/puerto físico con switch virtual/puerto virtual donde se encuentran conectados los hosts.

- Descarte de mensajes originados o destinados a dispositivos de red que no están comprendidos en la red física o virtual.

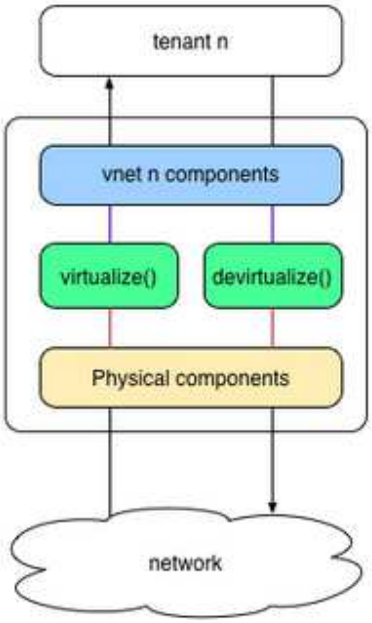


Figura 26 – Proceso de virtualización y desvirtualización [46]

3.ATRIBUTOS CARACTERÍSTICOS DE LAS ESTRATEGIAS DE VIRTUALIZACIÓN EN REDES DEFINIDAS POR SOFTWARE

Tal como se expresó en 1.DEFINICIÓN DEL PROBLEMA, el presente trabajo se centra en el estudio de soluciones de virtualización con vistas a las redes de datos de los centros de cómputo. Se focaliza en este caso de uso porque parece ser el de mayor complejidad al momento determinar los atributos que se le demanda a una solución de virtualización.

A. ATRIBUTOS CARACTERÍSTICOS

1) Transparencia

La transparencia implica que la capa de virtualización debe ser invisible tanto para el controlador como para los switches, permitiendo que su implementación no requiera modificaciones de diseño en los dispositivos de red existentes.

El desacople entre el diseño de los dispositivos de red y la tecnología de virtualización implica:

- la independencia de la tecnología de virtualización respecto de los dispositivos de red
- facilita los procesos de prueba en redes reales (de manera contraría podría darse el caso de que un dispositivo de red deba ser modificado y funcione únicamente en un entorno virtualizado).

La transparencia se basa en la definición de una interfaz adecuada entre la capa de virtualización y los dispositivos de red.

2) Aislación

La solución de virtualización debe permitir una fuerte aislación entre cada una de las redes virtuales.

Una completa aislación en redes SDN se debe dar en tres aspectos: el plano de control, el plano de datos y en los esquemas de direccionamiento.

i. *Plano de control*

Cada controlador debe controlar únicamente a su instancia de red virtual sin interferir sobre otras, administrando únicamente los flujos asociados al segmento de red correspondiente.

ii. *Plano de datos*

Los recursos físicos de los switches y de la red, se deben poder asignar a cada instancia virtual de manera tal de garantizar un determinado nivel de performance, evitando que una interfiera en el rendimiento de otra.

Los recursos principales a reservar son:

- Recursos físicos de los dispositivos de red

Procesamiento (CPU) y memoria (TCAM) de cada uno de los dispositivos de red intervinientes en cada red virtual.

- Ancho de Banda

Asignar a cada red virtual una fracción del ancho de banda disponible en los sucesivos enlaces de la red física. Para ello es necesaria la definición de una primitiva que permita dicha división (Ej. En WDM la asignación de una determinada longitud de onda λ).

- Tráfico

Asociar distintos tipos o patrones de tráfico a una o más redes virtuales, garantizando la aislación entre ellos.

iii. *Esquema de direccionamiento*

La aislación en el ámbito de los esquemas de direccionamiento, se refiere a la capacidad de que dos redes virtuales compartan el mismo esquema de direcciones. La premisa es que no se produzcan conflictos en las decisiones de *forwarding* de cada switch, que puedan ocasionar que el tráfico correspondiente a una red virtual circule por otra. Una de las formas de lograr aislación en este nivel es segmentando el espacio de flujos [49]. Otra solución consiste en asignar un *tag* o ID con el cual se encapsula el tráfico correspondiente a una determinada VN (Virtual Network) al atravesar un enlace compartido (ej. MPLS) [50].

La aislación a nivel de esquemas de direccionamiento, tiene una fuerte consecuencia en las capacidades de una solución para aislar el tráfico correspondiente a cada una de las redes virtuales.

3) Administración centralizada de políticas

Consiste en que la solución de virtualización posea una única interfaz para la configuración y administración de las políticas de todas las instancias de red virtuales.

4) Políticas extensibles

Las políticas definidas para cada red virtual deben poder ser modificadas de manera flexible ante un requerimiento por parte del usuario.

5) Reconfiguración en caliente

Permite la modificación de las políticas asociadas a una red virtual en tiempo de ejecución, minimizando las posibles disrupciones.

6) Abstracción de la topología de red física

Se refiere a la capacidad de que elementos de la red virtual (switches virtuales, enlaces virtuales, etc) no se correspondan necesariamente uno a uno, con sus respectivos físicos. Ello implica, por ejemplo, que un enlace virtual se pueda mapear con N enlaces físicos, así como también un switch virtual pueda estar comprendido por componentes de múltiples switches físicos (*big switch*) [51].

Esta característica permite la independencia topológica de las redes virtuales instanciadas respecto de la correspondiente a la red física subyacente.

7) Automatización en el despliegue

Se refiere a la capacidad de configurar instancias de red virtuales de manera automatizada, agilizando el aprovisionamiento y minimizando los tiempos de puesta en servicio de las mismas.

8) Alta disponibilidad y Resiliencia

Resiliencia es la capacidad inherente a una red de recuperarse ante una falla y así volver rápidamente a su funcionamiento normal, minimizando los tiempos de disrupción.

La resiliencia de una red está asociada a su configuración y diseño posibilitando que la misma sea tolerante a fallas, maximizando así su disponibilidad.

9) Grabar el estado de configuración de una red (*Snapshot*)

Se refiere a la capacidad de poder guardar el estado de configuración de una o más redes virtuales en un determinado instante de tiempo. En caso de una modificación en los parámetros de funcionamiento de manera deseada (ej. testeo de una nueva topología) o indeseada (ej. problemas de funcionamiento) se deberá poder volver a una configuración anterior, a través de la carga de la misma en forma instantánea por parte del administrador.

10) Multitenancy

Permite la utilización de una red física por parte de distintos grupos de usuarios, sin la posibilidad de que se percaten del uso compartido.

11) Virtualización recursiva

Es la capacidad de asignar una instancia de red virtual a otro hipervisor. En esta condición el hipervisor recursivo podrá virtualizar nuevamente dichos recursos y

asignarlos a diversos controladores u a otro hipervisor, para que realice nuevamente este procedimiento.

12) Degradación de performance (*Overhead*)

La incorporación de la "capa de virtualización" entre los planos de control y de datos, puede generar efectos negativos en la performance de la red. Ello se debe en general a OH (Overhead) o latencia, que en el contexto de redes SDN, se puede dar en ambos planos.

En el plano de datos se refiere a la adición de un encabezado propio, este es el caso del protocolo 802.1Q que añade un campo en el *header* de la trama ETH (Ethernet) disminuyendo el *throughput* de la red y causando latencia.

En cambio, tal como se expresó en el capítulo II. MARCO TEÓRICO apartado 1. SDN, el cambio de arquitectura que incorpora este nuevo paradigma supone tráfico entre el controlador y los dispositivos de red. En este caso, el agregado de una capa adicional "intermedia" puede generar degradación de performance y en consecuencia latencia.

Este fenómeno no se acota únicamente a las soluciones de virtualización de redes, sino que se manifiesta también en las soluciones de virtualización de cómputo. Ello se presenta en "Diagnosing Performance Overheads in the Xen Virtual Machine Environment" [52] y en "A comparison of virtualization technologies for HPC"[53].

13) Retrocompatibilidad con redes legacy

Posibilita la implementación de una solución de virtualización, manteniendo el diseño de red tradicional, sin implicar el recambio o incorporación de nuevo hardware.

La retrocompatibilidad con redes legacy requiere que la solución de virtualización y los protocolos necesarios para su implementación sean compatibles con el equipamiento de networking (switches/routers) tradicionales.

14) Funcionamiento en redes híbridas

Posibilita la implementación de soluciones de virtualización basadas en SDN a través de la incorporación de hardware compatible OF, pero manteniendo el diseño y la infraestructura de red existente (redes legacy) en un esquema de funcionamiento híbrido (tradicional/SDN) [54]. Ello permite obtener algunas de las ventajas que ofrece SDN, expuestas en el capítulo II. MARCO TEÓRICO, sin requerir la migración de la totalidad de la red.

15) Live Migration

Los centros de cómputo de la actualidad se caracterizan por poseer en su gran mayoría servidores virtuales, tal como se muestra en la Figura 13.

Una de las características principales de la virtualización de servidores, es que permite la migración de una VM hacia otro HW, sin disrupciones en el servicio. Ésta funcionalidad se denomina en general "*live migration"* y es una herramienta fundamental en los *datacenter* actuales.

Sin embargo, en ocasiones existe un acoplamiento importante entre las VMs y la configuración de red subyacente. Esto se da principalmente ya que las aplicaciones actuales, requieren la interacción entre distintas bases de datos y servidores para generar una respuesta de cara al usuario. Esta situación, supone tráfico predominante del tipo este-oeste tal como se expresó la sección correspondiente a redes de datos tradicionales del capítulo II. MARCO TEÓRICO.

Esto genera un contexto en el cual la migración de una o más VMs requiere también la migración de la configuración de red correspondiente. Esto permite mantener la interacción entre las aplicaciones, sin afectar la respuesta al usuario. A su vez, la característica de las VMs de poder ser migradas sin disrupción, impone que este proceso a nivel de *networking*, no implique tampoco interrupción del servicio.

Una solución de virtualización debe permitir la migración de una red virtual o parte de ella, mapeando la topología existente en nuevos switches [55] sin disrupción en el servicio o minimizándola. Debe garantizar además que los dispositivos de red conserven la configuración existente en sus tablas de flujo [56].

Este atributo es fundamental para posibilitar arquitecturas de datacenter híbridas, a través de las cuales se puedan integrar servicios en la nube o *cloud*.

16) Microsegmentación [57]

Actualmente los centros de cómputo cuentan con una arquitectura de seguridad perimetral, la cual está centrada en HW específico (Ej. Firewalls, IPS). Sin embargo, tal como se expresó en la sección correspondiente a redes de datos tradicionales del capítulo II. MARCO TEÓRICO, los patrones de tráfico han cambiado.

Esta situación conlleva a que los esquemas tradicionales de seguridad comiencen a ser obsoletos e inadecuados, requiriendo el filtrado de todo el tráfico entre servidores y aplicaciones.

La capacidad de realizar filtrado de tráfico de manera granular dentro de un centro de cómputos se denomina Microsegmentación [58]. Dicha funcionalidad se sustenta en la capacidad de segmentación que se obtiene a partir de la virtualización de la red.

4. SIMULACIÓN Y CARACTERIZACIÓN DE LAS ESTRATEGIAS DE VIRTUALIZACIÓN ABORDADAS

El presente trabajo de tesis no se acota sólo al relevamiento bibliográfico y a la propuesta de atributos considerados más relevantes para virtualizar las redes, sino que aporta valor agregado a través de la realización de nuevas simulaciones que permiten contrastar y/o enriquecer los resultados existentes.

Para la realización de las simulaciones propuestas se utilizó como herramienta Mininet [59]. Mininet permite la creación de *hosts*, switches, controladores y *links*, lo cual permite la configuración y emulación de redes SDN. Los hosts emulados implementan un kernel de Linux, mientras que los switches son compatibles con OF. Es así que los primeros pueden correr aplicaciones estándar, mientras que los segundos pueden interoperar con cualquier controlador compatible con OF 1.0 y 1.3 disponible en el mercado.

Mininet utiliza virtualización y *namespaces*, lo cual posibilita la creación de múltiples hosts y switches cada uno con sus propias interfaces de red, tablas de ruteo y tablas de ARP (Address Resolution Protocol), en una misma instancia de sistema operativo. Tal

como se introdujo en el apartado correspondiente a Virtualización de Servidores del capítulo II. MARCO TEÓRICO, este tipo de virtualización introduce poco OH. Ello permite realizar prototipado rápido de grandes redes en una PC *standalone* [60], lo cual convierte a Mininet en una herramienta muy potente y de basta utilización para investigación, desarrollo y enseñanza de SDN.

En el ANEXO 1: Herramientas utilizadas en la simulación, se listan la totalidad de herramientas utilizadas.

A. DESCRIPCIÓN DE LA SIMULACIÓN

Se expondrán los detalles del escenario de simulación propuesto y las distintas experiencias realizadas. Estas se utilizan para probar las funcionalidades de las estrategias de virtualización abordadas, como así también verificar la forma de implementación y el grado de cumplimiento de los atributos característicos presentados en 3. ATRIBUTOS CARACTERÍSTICOS DE LAS ESTRATEGIAS DE VIRTUALIZACIÓN EN REDES DEFINIDAS POR SOFTWARE.

1) Escenario de red simulado

Se diseñó un escenario de red que representa un centro de cómputos. Dicho caso de uso es el presentado en 1.DEFINICIÓN DEL PROBLEMA apartado b. Marco de trabajo y es el contexto sobre el cual se probarán las estrategias de virtualización abordadas.

La red simulada constituye la porción correspondiente al centro de cómputos de la red corporativa presentada en la Figura 27. Dicha red consta de dos sitios, una oficina central y el *datacenter*.

El *core* fue diseñado utilizando cuatro switches de capa 3 los cuales están interconectados entre sí, formando una topología de tipo anillo. Los enlaces entre los mismos son de 10 Gbps.

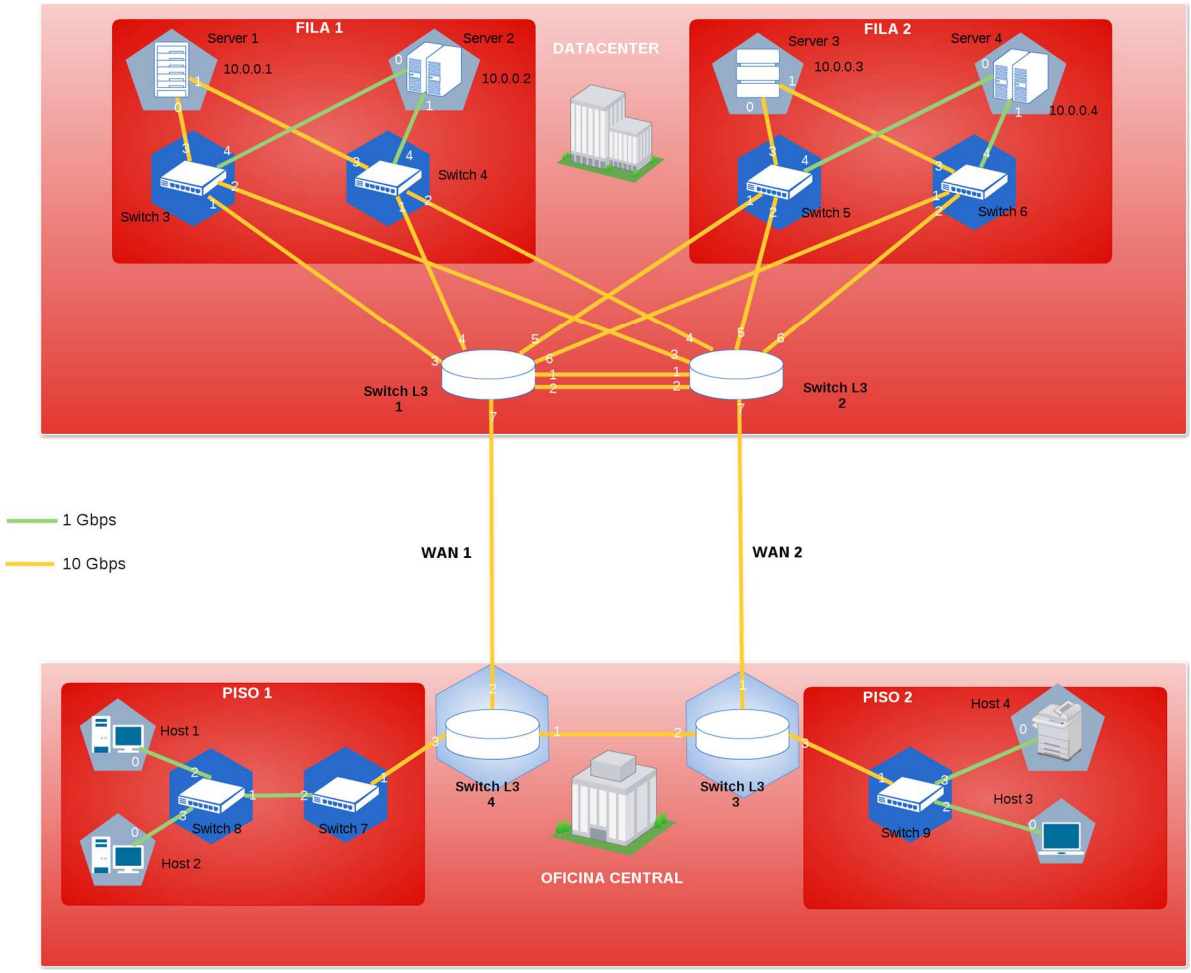


Figura 27 - Red corporativa

La red del centro de cómputos está diseñada con un switch *end of row* en cada una de las cuatro filas. Cada switch integra todas las conexiones de 1Gbps y 10 Gbps provenientes de la misma. Estos están representados mediante los switches 3,4,5 y 6. A su vez los *uplinks* de los mismos convergen en los dos switches de core (*layer 3*) 1 y 2 presentados en la Figura 28.

De acuerdo al diseño de anillo, los switches de *core* 1 y 2 se conectan a través de un enlace de WAN (Wide Area Network) con los switches 3 y 4 respectivamente. Estos últimos se encuentran en el centro de cómputo de la oficina central.

Se debe destacar que la red simulada es una representación de una red corporativa real. Sin embargo, esta se simplifica en lo que a su simulación respecta con el objetivo de optimizar los recursos de la máquina en donde se ejecuta la simulación, facilitando además el análisis de los resultados. Para ello, se asumen las siguientes consideraciones:

- Los switches de capa 3 funcionan en capa 2, es decir sin realizar ruteo.

- La porción de la red correspondiente a la oficina central se simplifica mediante dos *hosts* conectados directamente a los switches de *core* del centro de cómputos.

Estas consideraciones además permiten, que la simulación se acote al caso de uso presentado: *datacenter*. El reemplazo de la oficina corporativa por los dos hosts se justifica en el hecho de que los switches de *core* se encuentran funcionando en capa 2 y no en capa 3 tal como lo requeriría la red presentada en la Figura 27.

El escenario resultante se presenta en la Figura 28.

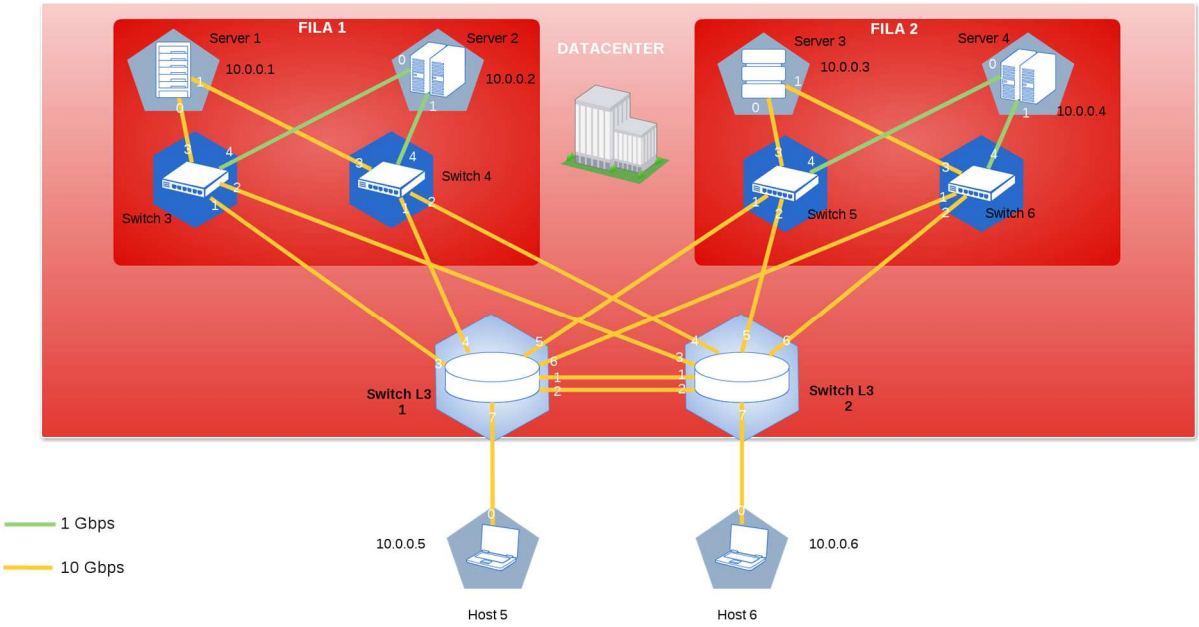


Figura 28 - Red simulada de un centro de cómputos

2) Componentes de la simulación

El presente apartado tiene por objetivo exponer el equipamiento y ambiente diseñado para realizar la simulación. Las herramientas utilizadas se presentan en el ANEXO 1: Herramientas utilizadas en la simulación.

La Figura 29 expone la configuración de dicho ambiente.

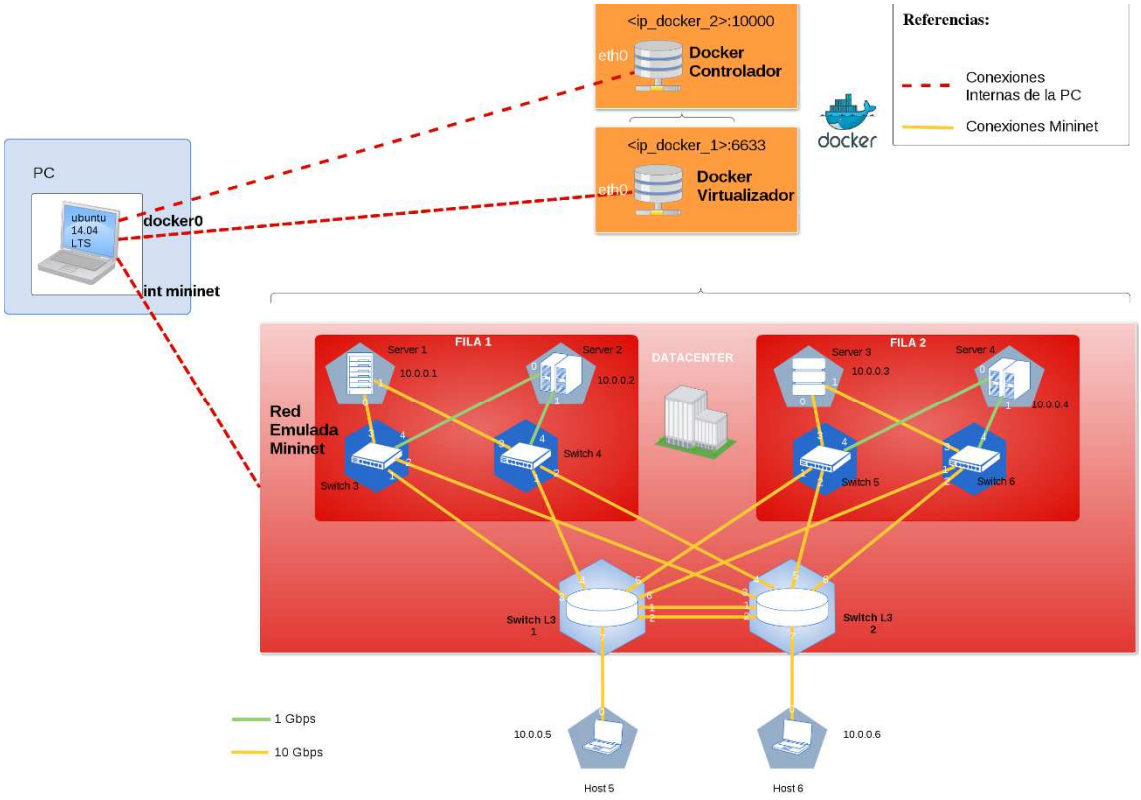


Figura 29 - Configuración y componentes de la simulación

Como se puede apreciar la simulación se realiza a través de una PC (Intel Core i5 4ta Generación, 16GB RAM, Disco SSD 256 GB) con OS Ubuntu 14.04 LTS. En dicha computadora se instala Mininet y se instancian Dockers [31], los cuales fueron previamente introducidos en el capítulo II. MARCO TEÓRICO apartado 2. VIRTUALIZACIÓN.

Inicialmente se habían generado múltiples máquinas virtuales tanto en VirtualBox como en VMware Player para los controladores y virtualizadores utilizados en el presente. Sin embargo, la asignación estática de recursos a cada VM y el OH asociado, agotan los recursos físicos de la PC generando un entorno no adecuado para la simulación.

Se utilizan Dockers como tecnología de virtualización ya que, como se expresó en el capítulo correspondiente, es la alternativa que genera menos OH. Debido a que toda la simulación se realiza en una PC convencional, este no es un aspecto menor en pos del óptimo aprovechamiento de recursos.

En este contexto se opta por diseñar contenedores para los virtualizadores y controladores utilizados. Cada *container* correrá únicamente una instancia de control o virtualizador según corresponda.

En el marco del presente trabajo, se generaron los siguientes contenedores. Los mismos se encuentran compartidos en *Docker Hub* [32], posibilitando la reutilización para futuros trabajos:

- mno808/floodlight:v0
- mno808/opendaylight:v3
- mno808/ryu:v1
- mno808/flowvisor:v2.0
- mno808/openvirtex:v3

Se configuran las NICs (Network Interface Card) virtuales en la PC para que los hosts y switches instanciados en mininet tengan conectividad con los contenedores utilizados en cada simulación.

En la Figura 30 se presenta el diagrama de capas análogo al expuesto en la Figura 19 - Arquitecura de red SDN virtualizada. El mismo permite exponer el rol de cada uno de estos componentes en las simulaciones propuestas.

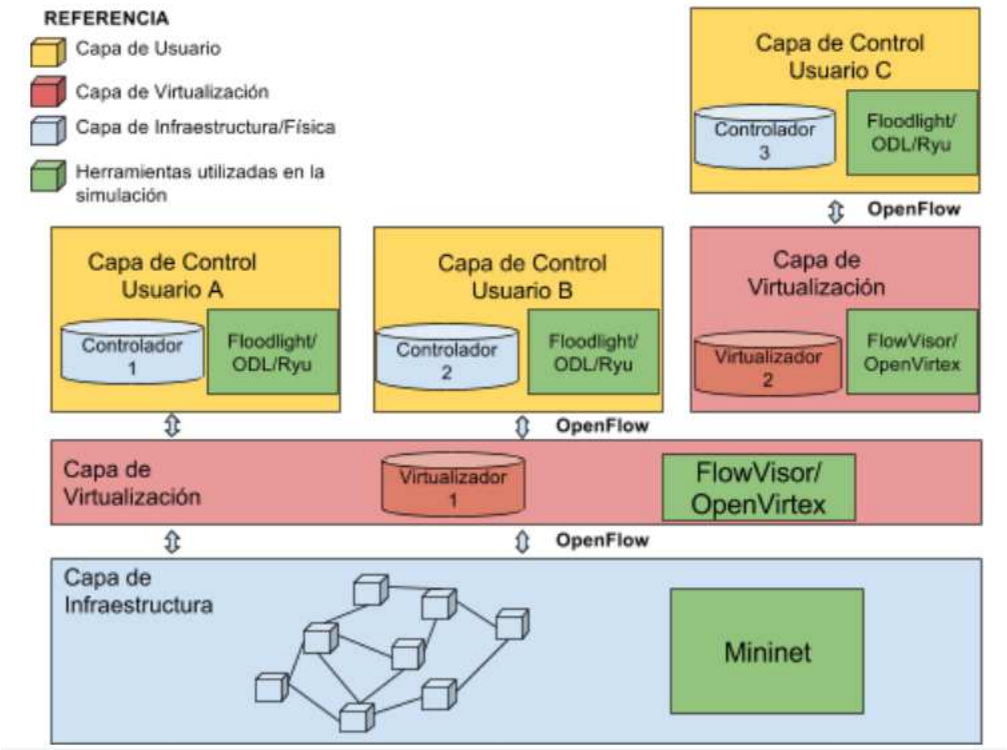


Figura 30 – Diagrama de capas: herramientas utilizadas y configuración de elementos para virtualizar una red SDN

i. *Capa de infraestructura*

La capa de infraestructura se simula a través de Mininet.

Mininet permite el diseño de topologías a medida e instanciación de las mismas. El diseño de una red implica escribir un script en Python, lo cual permite implementar el escenario propuesto (presentado en 1) Escenario de red simulado).

Para ello, se genera el código correspondiente (red_topo1.py). El mismo se presenta en el ANEXO 2: Script de la red emulada en mininet.

Para iniciar la simulación se debe ejecutar el siguiente comando:

```
sudo mn --custom red_topo1.py --topo redtopo --link tc --controller remote,ip=<ip_de_virtualizador/controlador> --mac
```

El script creado se instancia a través de los siguientes parámetros: --custom red_topo1.py --topo redtopo.

Tal como se expresó en el capítulo II. MARCO TEÓRICO apartado 1. SDN, el diseño de una red implica la existencia de un controlador, es por ello que al iniciar la simulación con Mininet se debe asignar dicha capa de control. Para ello, se utiliza el siguiente parámetro, indicando en este caso la IP remota correspondiente al mismo.

```
--controller remote,ip=<ip_de_virtualizador/controlador>
```

ii. *Capa de virtualización*

La capa de virtualización se implementa a través de Dockers. Se diseña un *container* para cada solución abordada: FlowVisor y OpenVirtex.

En el ANEXO 3: Configuración FlowVisor y ANEXO 4: Configuración OpenVirtex se exponen las consideraciones básicas para su funcionamiento. Los comandos expuestos en dichos anexos son los principales y por ello son transversales a todas las simulaciones.

iii. *Capa de control*

La capa de control se implementa también utilizando Dockers, a través de los cuales se ejecutan cada uno de los controladores utilizados: Floodlight y OpenDayLight.

En el ANEXO 5: Configuración OpenDayLight y ANEXO 6: Configuración FloodLight, se exponen las configuraciones básicas correspondientes.

B. CARACTERIZACIÓN DE ATRIBUTOS EN BASE A LOS RESULTADOS DE LA SIMULACIÓN

En el presente apartado se expondrán las características de diseño propias de cada una de las simulaciones realizadas, así como los resultados obtenidos para cada caso. Recordemos que las mismas tienen por objetivo poder definir el grado de cumplimiento de los atributos presentados en 3. ATRIBUTOS CARACTERÍSTICOS DE LAS ESTRATEGIAS DE VIRTUALIZACIÓN EN REDES DEFINIDAS POR SOFTWARE, tanto para FV como OVX.

1) Transparencia

i. *Procedimiento*

Se instancia la red del centro de cómputos presentada en la Figura 28 a través de Mininet. Luego se le asigna el controlador. Se procede a verificar que tienen control sobre la totalidad de la red emulada y en consecuencia es posible visualizar dicha topología y establecer comunicación entre los hosts.

En forma posterior, se procede a virtualizar una porción de la red y se asigna dicha instancia a uno de los controladores. Este proceso se realiza sin modificar ninguna variable de configuración, tanto en la capa de control como en los dispositivos subyacentes. Esta condición es la que permitirá afirmar, si las soluciones de virtualización a tratar son transparentes. Para ello, se debe verificar que el controlador pueda efectivamente visualizar la topología de red virtual y los equipos correspondientes.

Este procedimiento permite comprobar la transparencia para ambas soluciones de virtualización: FlowVisor y Openvirtex.

Procedimiento resumido:

- Simular la red del centro de cómputo a través de Mininet
- Asignar un controlador
- Visualizar la topología de red a través de la interfaz del controlador
- Virtualizar una porción de la red y asignársela a un controlador, sin realizar modificaciones en la configuración de éste, ni en los dispositivos subyacentes.
- Visualizar la topología de red virtual desde la interfaz de administración del controlador.

El desarrollo de la simulación se presenta en ANEXO 7: Simulación de atributo de transparencia.

ii. *Resultado*

Se logra la virtualización de la red emulada utilizando ambas soluciones (FV y OVX).

La incorporación de la capa de virtualización no requirió en ninguno de los casos, la modificación de la configuración de los elementos de la capa física ni tampoco de la de control. En consecuencia, se puede afirmar que ambas soluciones resultan ser **transparentes.**

2) Aislación

Plano de control

i. *Procedimiento*

Se simula la red del centro de cómputos presentada en la Figura 28 a través de Mininet. A partir de esta, se configuran dos redes virtuales.

Para verificar la aislación a nivel del plano de control, se procede a asignar cada red virtual a dos controladores distintos. A partir de ello, se podrá verificar que efectivamente cada instancia de control administra únicamente la red virtual que le fue asignada. Para ello, se debe acceder a la interfaz de administración de cada uno y verificar esta condición.

Este proceso se realiza tanto en el caso de FlowVisor como OpenVirtex.

Procedimiento resumido:

- Simular la red del centro de cómputo a través de Mininet.
- Configurar dos redes virtuales.
- Asignar el control de ambas redes virtuales a dos instancias de control distintas.
- Visualizar la topología de red desde ambos controladores.
- Verificar que cada controlador visualice y en consecuencia tenga facultades de administrar únicamente la red virtual bajo su control.

El desarrollo de la simulación se presenta en ANEXO 8: Simulación de atributo de aislación.

ii. *Resultado*

Se logra virtualizar la red emulada y configurar dos instancias virtuales. Se consigue asignar cada red virtual a su correspondiente controlador. Ambas redes y sus componentes se visualizan de manera correcta desde las interfaces de cada uno.

En forma posterior se realizan pruebas de comunicaciones entre hosts correspondientes a la misma red virtual a través de pings entre ellos. En todos los casos se logra una correcta respuesta.

Se repite esta prueba, pero entre hosts correspondientes a diferentes redes virtuales, verificando la imposibilidad de comunicación entre ellos.

De esta manera se verifica la correcta **aislación del plano de control** para ambas soluciones de virtualización.

Plano de datos

i. *FlowVisor*

A. Aislación de la CPU del switch

La sobrecarga de CPU en los switches se da por diversas causas [33]. Se presentan a continuación las dos más representativas:

> Nuevos flujos
> FV registra la tasa de arribos de paquetes que requieren nuevas reglas de flujo a través de la intercepción de dicho tipo de mensajes (Sección 5.3.3 del estándar OF 1.0 [13]). Si estos mensajes superan un determinado umbral, FV inserta en el switch una regla para que dichos paquetes sean descartados por un intervalo de tiempo. Se debe recordar que esto es posible ya que FV se sitúa lógicamente entre el switch y el controlador.
>
> Solicitudes del controlador (ej. editar reglas de flujos, solicitud de estadísticas)
> FV limita la tasa de mensajes OF a un determinado valor umbral. La limitación en este sentido se encuentra en el hecho de que los mensajes OF consumen recursos de CPU de acuerdo al tipo de mensaje y al HW (Hardware) sobre el cual se ejecutan. FV no tiene ninguna herramienta que permita considerar esta variable.

Tal como se puede apreciar, FV implementa mecanismos de aislación de CPU para permitir múltiples redes virtuales sobre los mismos equipos de red físicos. Sin embargo, *OF 1.0 no contempla un mecanismo que permita de manera nativa la abstracción de este tipo de recursos*, con lo cual esta funcionalidad escapa a la concepción de diseño de FV y se presenta como un *work-around* para minimizar esta problemática [33].

B. Aislación del BW

OpenFlow 1.0 no permite de manera nativa administrar QoS (Quality Of Service) (Sección 5.3.4 “Queue Configuration Messages” del estándar OF 1.0 [13]). Al igual que en el caso de aislación de CPU, FV implementa un mecanismo no nativo de OF para lograr aislación de BW.

Para ello utiliza los 3 bits de PCP (Priority Code Point) correspondientes al encabezado de VLAN, los cuales permiten asignar de manera estándar a una trama hasta 8 niveles de prioridad.

Debido a que OF expone primitivas para manejar VLANs y en consecuencia los bits de PCP, FV prioriza el tráfico de cada red virtual asignando un valor en este campo a cada paquete que se corresponde con las entradas de flujo de dicho slice. Esto se realiza a través de una acción del tipo “*set VLAN priority*” en caso de coincidencia.

C. Aislación TCAM/Entradas de flujo

FV contabiliza el número de entradas de flujo por slice asegurándose de que no excedan el límite prestablecido.

OF cuenta con un comando que permite conocer el número de entrada de flujos en un switch. Cuando un controlador asociado a una red virtual excede el límite de entradas de flujo, ante cualquier nueva regla recibirá un mensaje de error (Sección 5.4.4 del estándar OF 1.0 [13]) indicando que la tabla se encuentra llena.

En FlowVisor: A Network Virtualization Layer [33] se exponen resultados de pruebas realizadas al respecto de los mecanismos de aislación expuestos.

D. Aislación de tráfico

a. Procedimiento

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet. A partir de esta se configuran dos redes virtuales.

Se diseña una simulación en la cual se asigna el control de todo el tráfico entre dos hosts a un controlador, mientras que se asigna a un segundo controlador sólo el tráfico TCP (Transmission Control Protocol) asociado al puerto 555. Se enciende y se apaga cada uno de los controladores en pos de verificar que ambos tienen capacidad de administración del mismo tipo de tráfico.

Este proceso se realiza para el caso de FV.

Procedimiento resumido:

- Simular red del centro de cómputo a través de Mininet.
- Configurar dos redes virtuales entre dos hosts.
- Asignar el control de todo el tráfico L2, L3 y L4 a un primer controlador y sólo el tráfico TCP correspondiente al puerto 555 a un segundo controlador (por definición ambos controladores tendrán facultades de administración del tráfico TCP puerto 555).
- Apagar el primer controlador y verificar que es posible cursar tráfico TCP en el puerto 555 entre ambos hosts. Luego realizar lo mismo con el segundo.

El desarrollo de la simulación se presenta en ANEXO 9: Simulación de atributo de aislación de tráfico (Plano de datos).

b. Resultado

Se demuestra que el mismo “tipo” de tráfico puede ser compartido por dos o más controladores, quienes tendrán la capacidad de administración sobre el mismo.

Esta situación no garantiza la aislación del tráfico asociado a cada red virtual, sino que, por lo contrario, permite concluir que dos o más redes virtuales pueden controlar el mismo.

Tomando como complemento lo expuesto en el ANEXO 8: Simulación de atributo de aislación apartado **Aislación de los esquemas de direccionamiento** B.FlowVisor, la posibilidad de superposición en la definición de flujos asociados a cada red virtual complejiza el escenario para asegurar la correcta aislación a nivel de tráfico. Esto prueba

lo enunciado en [33] (Apartado Virtual Address Space) donde se especifica que si dos redes virtuales comparten el mismo espacio de flujos no se puede garantizar la correcta aislación.

ii. *OpenVirtex*

Tal como se expresa en “OpenVirteX: Make Your Virtual SDNs Programmable” [43], OVX permite la creación de redes virtuales con la posibilidad de utilizar la totalidad de los espacios de flujo en cada una de ellas. Esto viabiliza la definición de reglas superpuestas en las redes virtuales creadas.

En este sentido, se puede afirmar una ventaja comparativa de OVX frente a FV al respecto de la aislación de tráfico. En lo referido específicamente a esquemas de direccionamiento la capacidad de superposición se prueba en 8. ANEXO 8: Simulación de atributo de aislación apartado A. OpenVirtex.

En lo relacionado a QoS (Aislación de la CPU, BW y TCAM) OVX expone las mismas limitaciones que i.FlowVisor, ya que ambos utilizan OF 1.0. Tal como se expuso anteriormente, esta versión del estándar no incluye en su definición primitivas que permitan garantizar calidad de servicio (Sección 5.3.4 “Queue Configuration Messages” del estándar OF 1.0 [13]).

Esquema de direccionamiento

i. *Procedimiento*

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet. A partir de ello se configuran dos redes virtuales.

Se procede a asignar las mismas direcciones IP a los *hosts* correspondientes a ambas instancias, para forzar la superposición en la red emulada. Luego se procede a realizar pruebas de pings en forma simultánea entre los *hosts* que comparten las mismas direcciones.

Este proceso se realiza tanto en el caso de FlowVisor como OpenVirtex.

Procedimiento resumido:

- Asignar las mismas direcciones de IP a los respectivos hosts de la red gris y naranja.
- Realizar un ping en forma simultánea en las dos redes virtuales entre hosts con las mismas direcciones IP.
- Verificar que no se generan inconvenientes en la comunicación e identificar el proceso asociado para ello.

El desarrollo de la simulación se presenta en ANEXO 8: Simulación de atributo de aislación.

ii. *Resultado*

En el caso de OVX, los switches de borde modifican las direcciones de IP origen y destino dentro de la red virtual. Luego de que los paquetes son direccionados según corresponda dentro de la misma, otro switch de borde tiene la responsabilidad de volver a asignar las direcciones originales para que éstos sean entregados a los hosts de destino según corresponda. Ello permite la superposición de esquemas de direccionamiento en dos redes virtuales corriendo sobre la misma red física.

En contraposición FV no asegura una correcta aislación al respecto de los esquemas de direccionamiento ya que no posee ningún mecanismo de reescritura similar al que implementa OVX. En este contexto no se puede asegurar que no se superpongan los espacios de flujo en toda la topología de la red. Tal como se expresa en [33], no se puede garantizar la aislación y en consecuencia la superposición de direcciones IP en dos instancias virtuales de la red.

3) Administración centralizada de políticas

i. *FlowVisor*

FV cuenta con una API [61] que permite la configuración y el monitoreo de las redes virtuales generadas. Dicha API se basa en la especificación JSON RPC 2.0 [62].

La misma se accede desde el servidor donde se encuentra instalado FV, a través del comando *fvctl* tal como se muestra a continuación:

```
# fvctl /dev/null --version
fvctl-0.1
```

Dicha API permite la administración y monitoreo centralizado de todas las redes virtuales generadas a través de una determinada instancia de FV. Si se ejecuta el siguiente comando, se pueden visualizar todas las funcionalidades de administración y monitoreo, como así también la forma de utilizarla.

```
# fvctl /dev/null --help
```

La salida con el detalle de parámetros se expone en el ANEXO 10: APIs de administración. Adicionalmente, se incluye una breve descripción de cada uno de ellos. En la documentación de FV [61] se encuentra la especificación formal de cada uno de los comandos expuestos.

ii. *OpenVirtex*

OpenVirtex define dos APIs: la primera cuya funcionalidad es el monitoreo (*Monitoring* API); mientras que las segunda (*Tenant* API) se utiliza para obtener información y a su vez configurar las redes virtuales.

Ambas APIs se encuentran centralizadas y son las responsables de la gestión de todas las redes virtuales creadas a través de OVX. Se acceden de manera local dentro del directorio de instalación de OpenVirtex:

```
#~/OpenVirtex/utils/ovxctl.py
```

En dicha ruta, si se ejecuta el siguiente comando, se pueden visualizar las funcionalidades disponibles:

```
# python ovxctl.py –n –help
```

La salida con el detalle de parámetros de administración y monitoreo se exponen en el ANEXO 10: APIs de administración. Adicionalmente, se incluye una breve descripción de cada uno de ellos.

En la documentación de OVX [48], se encuentran la especificación formal de los comandos expuestos.

4) Políticas extensibles

El procedimiento y los resultados relacionados con el atributo políticas extensibles, se exponen en 5) Reconfiguración en caliente.

5) Reconfiguración en caliente

i. *Procedimiento*

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet. Se genera una red virtual y se asigna a un controlador.

Luego se procede a modificar su configuración en pos de extender la conectividad de la misma a diversos hosts. Dicha modificación de políticas se realiza en caliente, es decir mientras la red está brindando servicio. En este contexto se busca determinar si al realizar la modificación de una política, la misma se puede realizar en caliente sin implicar ningún tipo de disrupción o afectación del servicio.

El procedimiento presentado permite comprobar este atributo para ambas soluciones de virtualización: FlowVisor y Openvirtex.

Procedimiento resumido:

- Simular la red del centro de cómputo a través de Mininet.
- Crear una red virtual y asignar a un controlador.
- Extender la conectividad de la red virtual a diversos hosts mientras está prestando servicio.
- Verificar que dicho proceso se puede realizar en caliente sin disrupción.

El desarrollo de la simulación se presenta en ANEXO 11: Simulación de atributos: Políticas extensibles y Reconfiguración en caliente.

ii. *Resultado*

A. OpenVirtex

OVX resulta una solución que tiene versatilidad en cuanto a la extensión de sus políticas de red, permitiendo modificar las mismas mientras la red está prestando servicio. Una característica de interés es que las actualizaciones se reflejan de manera inmediata, minimizando las posibilidades de afectación de los servicios de cara al usuario (reduciendo posibles tiempos de disrupción).

B. FlowVisor

FV resulta una solución que permite la extensión de sus políticas, posibilitando además modificar las mismas mientras las redes virtuales permanecen en servicio. Sin embargo, en ocasiones dichas actualizaciones no se vieron reflejadas a nivel del controlador**.** *Esta situación se dio en forma aleatoria, no pudiendo determinar las causas que lo originan.*

6) Abstracción de la topología de red física

i. *Procedimiento*

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet.

Luego se proceden a diseñar dos redes virtuales cuyo objetivo es permitir la conectividad entre dos hosts/servidores de la red.

La primera red se diseña utilizando un switch virtual para interconectar los dos hosts, mientras que la segunda se desarrolla utilizando links virtuales.

De esta manera se buscará comprobar la capacidad de la solución de virtualización para generar *virtual links* que permitan la abstracción de N enlaces físicos, como así también la definición de switches virtuales que embeban componentes de múltiples físicos.

Procedimiento resumido:

- Simular red del centro de cómputo a través de Mininet.
- Diseñar una red utilizando un switch virtual que permita la abstracción de múltiples switches físicos.
- Visualizar la topología de red resultante a través de la interfaz web del controlador.
- Diseñar una red utilizando un *virtual link* que permita la abstracción de múltiples enlaces físicos.
- Visualizar la topología de red resultante a través de la interfaz web del controlador.

El desarrollo de la simulación se presenta en el ANEXO 12: Simulación de atributo de Abstracción de la topología de red física.

ii. *Resultado*

A. OpenVirtex

OVX permite el diseño de redes virtuales utilizando tanto switches como *virtual links*. A través de ambos, es posible la generación de abstracciones de la red que permitan ocultar la configuración física, exponiendo así, una nueva topología de cara al usuario.

B. FlowVisor

FV no permite la generación de redes virtuales en forma independiente de la topología física.

Esta solución se basa en la definición de espacios de flujo o *flowspaces* tal como se expresó en 2. NUEVAS ESTRATEGIAS DE VIRTUALIZACIÓN apartado a. FlowVisor, sobre la base de la topología que caracteriza a la red subyacente.

7) Automatización en el despliegue

i. *Procedimiento*

OVX cuenta con un módulo denominado "*embedder*" cuya funcionalidad es la de automatizar el despliegue de redes virtuales. Dicho componente permite únicamente dos topologías:

- *Bigswitch*: permite la creación de un switch virtual que representa una abstracción de múltiples switches físicos. Los *hosts* que intervienen en la red virtual, se conectan a través de él.
- Red física: permite la creación de una red virtual que representa a la red física 1:1, es decir que ambas topologías resultan idénticas.

La aplicación embedder presenta la limitación de no soportar topologías *custom*, es decir, combinaciones de las opciones anteriormente expuestas.

Para cualquiera de los dos tipos de despliegue permitidos, *embedder* requiere la descripción de la red virtual a través de JavaScript.

La simulación consiste entonces en diseñar el script correspondiente a la red virtual a automatizar en cuanto a su despliegue. Luego se procede a probar dicho proceso utilizando "*embedder*" a través de la instanciación del código desarrollado.

Procedimiento resumido:

- Describir la red a desplegar en JavaScript.
- Instanciar dicho script a través de la aplicación "embedder".

El desarrollo de la simulación se presenta en ANEXO 14: Simulación de atributo automatización en el despliegue.

ii. *Resultado*

A. OpenVirtex

OVX a través de la aplicación *embedder* permite el despliegue automatizado de redes virtuales.

Se utiliza la palabra "automatizado" puesto que, realizando modificaciones menores en el script, se puede desplegar de manera ágil una nueva red virtual siempre y cuando tenga una configuración similar. Adicionalmente se justifica en el hecho de que permite despliegues más rápidos si se compara con el uso de la API de administración; ésta última implicaría la escritura de los comandos de configuración en forma manual.

Sin embargo, se considera que dicho método de automatización no es flexible puesto que, si se requiere una topología de red distinta, es necesario escribir un nuevo script o realizar modificaciones mayores del mismo.

B. FlowVisor

FV no define ningún mecanismo que permita el despliegue de manera automatizada, ni tampoco contempla en su diseño un proceso que posibilite agilizar el aprovisionamiento de una red virtual.

8) Alta disponibilidad y Resiliencia

i. *Procedimiento*

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet. Se procede a diseñar una red virtual con el objetivo de permitir la comunicación entre dos *hosts*. Se contemplan en el diseño caminos redundantes entre ellos, considerando que en condiciones de funcionamiento normal sólo uno estará activo.

Una vez creada la red virtual, se realizan pruebas de tráfico entre los hosts para verificar el correcto funcionamiento de la misma. Adicionalmente, se analizan las tablas de flujo de los switches y se realiza *sniffing* del tráfico para determinar por qué camino se está cursando el mismo

Luego se procede a bajar el vínculo principal. Debido a que se trata de una red emulada esto se realiza a través de Mininet, aunque se genera el mismo efecto que el producido por un corte en el vínculo físico entre ambos.

Luego de ello, se debe verificar que la red sigue prestando servicio habiendo redirigido el tráfico hacia otro vínculo. Para comprobar esta condición, se proceden a analizar nuevamente las tablas de flujo y así validar que se modificaron en consecuencia. Una vez que se determina el nuevo camino activo, se procede a realizar *sniffing* del tráfico correspondiente.

Procedimiento resumido:

- Simular red del centro de cómputo a través de Mininet.

- Diseñar una red virtual que permita la comunicación entre dos hosts. Verificar el camino que toma el tráfico entre ellos a través del análisis de las tablas de flujo en los switches de la red.
- Sniffear el tráfico entre los dos hosts de manera tal de realizar una segunda verificación al respecto del camino que toma el tráfico.
- Establecer un camino alternativo a través de la interfaz de configuración de la solución de virtualización.
- Bajar el servicio del camino principal entre los dos hosts y verificar dicha condición.
- Verificar que la red siguió brindando servicio y que efectivamente el tráfico se enrutó de manera automática por el camino secundario.

El desarrollo de la presente simulación se presenta en el ANEXO 13: Simulación de atributo de alta disponibilidad/resiliencia.

ii. *Resultado*

A. OpenVirtex

Se logró verificar empíricamente el funcionamiento del mecanismo de alta disponibilidad/resiliencia que implementa OVX. Se pudo comprobar que, ante una condición en la cual el vínculo activo falla, la red virtual se reconfigura de manera automatizada para redirigir el tráfico por el vínculo de *backup*. En el contexto de la simulación realizada, dicho proceso no implicó la pérdida de ningún ping entre los hosts, siendo únicamente notorio un leve retardo en uno de ellos. Se infiere que este tiempo es el requerido para la actualización de las tablas de flujo en los switches y que efectivamente se comience a enrutar el tráfico por el camino redundante.

B. FlowVisor

FV no define ningún mecanismo que permita que la red se reconfigure de manera automática ante una falla. De esta manera se puede concluir en que FV no implementa

ninguna característica de valor agregado en cuando a resiliencia y alta disponibilidad, asociado a la virtualización de la red.

9) Grabar el estado de configuración de una red (Snapshot)

i. *Procedimiento*

A. OpenVirtex

OVX posibilita guardar las topologías de red virtuales configuradas. En la documentación asociada [63] se denomina a esta característica persistencia. Dicha solución de virtualización permite grabar la configuración de la red en una base de datos, lo cual permite que, ante cualquier contingencia la misma pueda ser restituida. OVX, utiliza como base de datos MongoDB [64].

Si bien a través de esta funcionalidad se pueden recuperar las topologías de red virtuales instanciadas, presenta la limitación de que no se restauran las configuraciones de las tablas de flujo en los switches correspondientes. En dicha condición las redes recuperadas no se encuentran en el mismo estado de funcionamiento, sino que lo hacen en el “inicial”.

B. FlowVisor

FV no cuenta con documentación disponible al respecto de la funcionalidad de *snapshot* [45]. En función de ello, para el diseño de la simulación y la evaluación de dicha funcionalidad, se procedió a analizar la API de administración en pos de identificar aquellos comandos que pudieran estar asociados a grabar el estado de configuración de la red.

A partir de dicho análisis se identificó el comando *fvctl save-config*, el cual de acuerdo a su descripción permitiría potencialmente grabar la configuración de las redes virtuales instanciadas. El diseño de la simulación propuesta se basa en la utilización de dicho comando.

El procedimiento de prueba utilizado para verificar ambas soluciones de virtualización se presenta a continuación:

Procedimiento resumido:

- Simular red del centro de cómputo a través de Mininet.
- Crear una red virtual y asignarla a un controlador.
- Verificar el correcto funcionamiento de dicha red y grabar su configuración en la base de datos.
- Interrumpir el servicio de OVX/FV y luego iniciarlo nuevamente.
- Verificar que la red virtual y su configuración asociada, se haya reestablecido luego de la restitución del servicio de la capa de virtualización.
- Verificar el funcionamiento de dicha red a través de pruebas de comunicación entre los hosts asociados a la misma y visualizar la topología resultante desde la interfaz del controlador.

El desarrollo de la presente simulación se presenta en el ANEXO 15: Simulación de atributo Grabar el estado de configuración de una red (Snapshot)

ii. *Resultado*

La funcionalidad simulada se limita a la posibilidad de recuperar las topologías de las redes virtuales instanciadas. Dicha limitación no permite denominar a esta funcionalidad "*snapshot*". Para ello, OVX debería posibilitar guardar no solo la configuración de la red virtual (en cuanto a su topología y componentes) sino también los flujos prexistentes en los switches.

Esto permite la recuperación de la red en su estado de funcionamiento "inicial" y no en el que se encontraba en forma previa a la interrupción del servicio. De esta manera se puede concluir en que OVX implementa éste atributo en forma parcial.

Para el caso de FV, la situación es análoga. Dicha solución permite “grabar” la topología de las redes virtuales configuradas, pero no así, el estado de los flujos previos a la interrupción del servicio.

10) Multitenancy

El atributo de *multitenancy* de ambas soluciones se demuestra a través de las simulaciones realizadas en torno al atributo de 2)Aislación - **Plano de control**.

A partir de la experiencia diseñada, se puede demostrar que dos o más usuarios pueden disponer de su propio controlador a partir del cual tienen capacidades de administración y configuración de sus propias redes virtuales. De esta manera, cada uno de los usuarios dispone de la gestión de parte de la red física, sin poder apreciar la condición de uso compartido.

11) Virtualización recursiva

i. *Procedimiento*

Se simula la red del centro de cómputos presentado en la Figura 28 a través de Mininet. Se proceden a crear dos redes virtuales. Una de ellas se asigna a un controlador, mientras que la otra a una segunda instancia de virtualización.

Luego se procede a virtualizar de manera recursiva la red a través del segundo hipervisor. La misma se asigna en forma posterior a un controlador.

Procedimiento resumido:

- Simular red del centro de cómputos a través de Mininet
- Se crean dos redes virtuales.
- Se asigna el control de una red a un controlador y el de la otra red a una segunda instancia del virtualizador.
- Se virtualiza en forma recursiva la red y se asigna a un controlador.

- Visualizar a través de la interfaz del controlador la red virtualizada recursivamente.

El desarrollo de la presente simulación se presenta en ANEXO 16: Simulación de atributo virtualización recursiva.

ii. *Resultado*

Ambas soluciones son comparables ya que permiten la virtualización recursiva de redes virtuales.

La ventaja que se aprecia en OVX al respecto de la virtualización recursiva se apoya en el atributo 6) Abstracción de la topología de red física. Este último permite que la red virtual a aprovisionar oculte los detalles de diseño y topología de la red subyacente, lo cual no es posible a través de FV.

12) Degradación de performance

i. *FlowVisor*

La adición de una capa lógica entre los planos de datos y control genera como consecuencia degradación en la performance del sistema. En este sentido, FV no agrega ningún OH en el plano de datos, a diferencia de IEEE 802.1Q por ejemplo, sino que se genera *delay* cuando existen mensajes entre los dispositivos de red y el controlador [33]. En OF por ejemplo, este caso se da cuando no existe una regla de flujo que coincida con el encabezado de un paquete entrante.

ii. *OpenVirtex*

Al igual que para el caso de FV, OVX genera degradación de la performance en el canal de control, debido al proceso de reescritura de mensajes OF entre los equipos de red y los controladores.

En cambio, OVX también genera degradación de performance en el plano de datos. Esto se da, por ejemplo, debido al proceso de reescritura de paquetes en los switches de borde las redes. Este proceso es el que posibilita la superposición del mismo esquema de direccionamiento en diversas redes virtuales, tal como se demostró en el apartado **Esquema de direccionamiento** correspondiente al atributo 2) Aislación .

iii. *Resultado*

Para la caracterización de la degradación de performance se tomará como referencia la evaluación expuesta en OpenVirtex: Make Your Virtual SDNs Programmable [43], la cual compara la latencia introducida por ambas soluciones de virtualización abordadas. El procedimiento de prueba se expone en el documento referenciado.

La Figura 31 expone el resultado de dicha prueba:

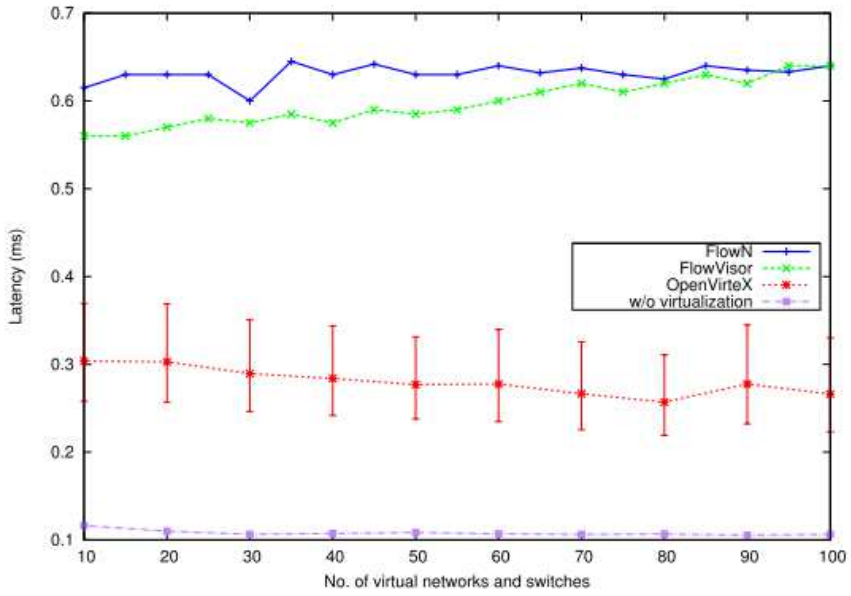


Figura 31 – Comparación de latencia introducida por FlowVisor y OpenVirtex en relación a la red sin virtualizar [43]

Ambas soluciones de virtualización, presentan una degradación de la performance en relación a no virtualizar la red. Sin embargo, OVX introduce una latencia sensiblemente menor al sistema, si se la compara con FV.

13) Retrocompatibilidad con redes legacy

De acuerdo a los expuesto en el apartado 2. NUEVAS ESTRATEGIAS DE VIRTUALIZACIÓN, tanto FV como OVX requieren para su funcionamiento, la interacción con los dispositivos de red a través del protocolo OF 1.0 [13]. A partir de dicha condición de diseño, no es posible la retro compatibilidad con redes tradicionales o legacy ya que los dispositivos de red carecen en general de compatibilidad con dicho protocolo.

14) Funcionamiento en redes híbridas

i. *Procedimiento*

Se diseña una red simplificada en mininet compuesta por un único switch y un *host*. La misma cuenta con una interfaz del dispositivo de red conectada (*bridged*) con eth0 de la PC. Esta servirá de medio para la comunicación con una red tradicional o legacy.

Se virtualiza la red emulada en mininet a través de FV y OVX asignando su control a una instancia de Floodlight. Se procede a realizar pruebas de ping entre el host emulado (Mininet) y los equipos conectados a la red *legacy*.

A. Topología de red diseñada

La Figura 32 presenta la topología de red utilizada para verificar si es posible que redes virtualizadas convivan con redes *legacy* (redes híbridas).

La topología de red emulada en mininet es más sencilla que la utilizada para probar el resto de atributos estudiados. Sin efecto de ello, los resultados son válidos para cualquier tipo de red SDN.

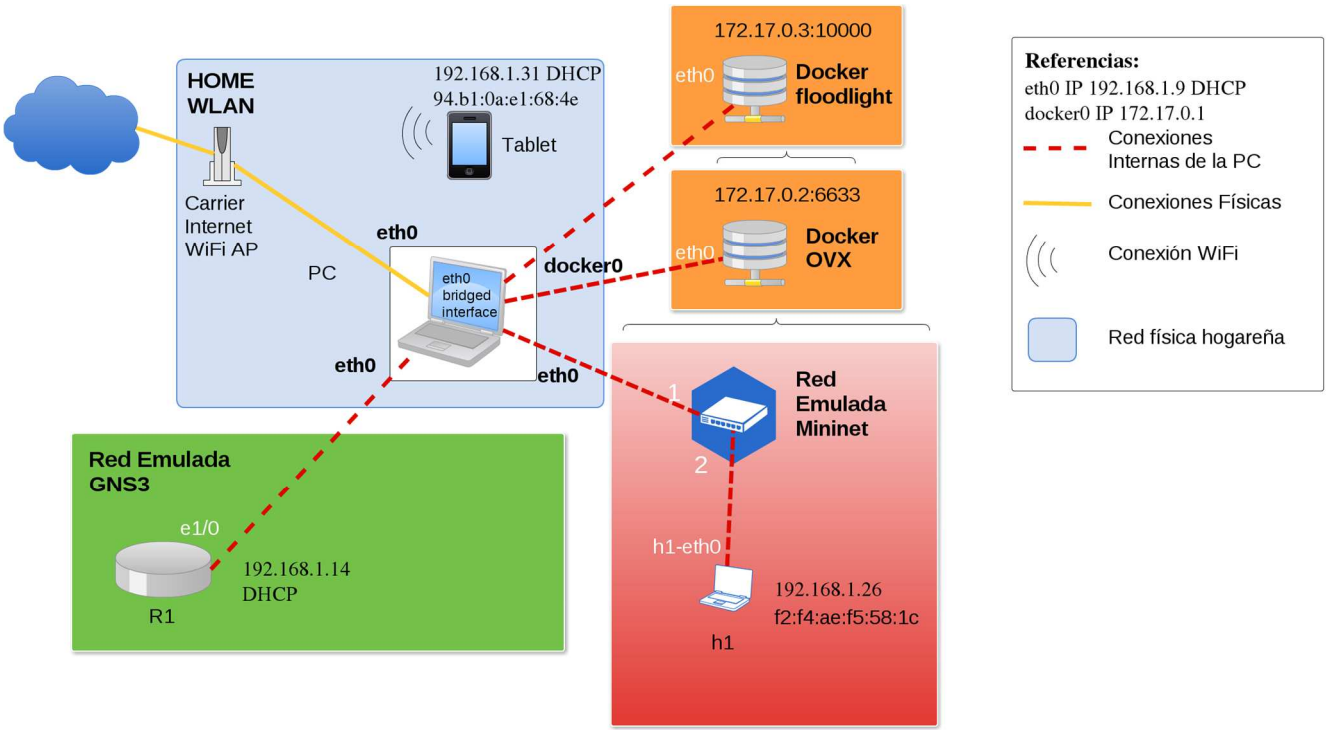


Figura 32 – Red híbrida

B. Descripción de la red

La red de la Figura 32 está compuesta por cinco partes:

- Red física (recuadro azul)
- Red emulada con GNS3 (recuadro verde)
- Red emulada con Mininet (recuadro rojo)
- Contenedores de docker (recuadros naranjas)

La PC se encuentra conectada a través de su NIC (interfaz eth0) al modem de acceso a internet (Sagecom CS5001), el cual sirve también de AP (Access Point) WiFi. Se configura un dispositivo móvil (tablet) y se conecta a la red inalámbrica. Tanto la PC como la tablet están configuradas para obtener su dirección IP a través del servidor DHCP (Dynamic Host Configuration Protocol) del proveedor de servicios de internet.

En paralelo, en la PC se configura una red emulada utilizando GNS3. La misma incorpora un router Cisco 7200, el cual se conecta a través de su interfaz e1/0 a la interfaz física eth0.

Se configura una red OpenFlow mediante Mininet, la cual está constituida por un host y un switch OVS (Open vSwitch). Dicho switch tiene dos interfaces: la eth2 conectada al host; y la eth1 conectada a la eth0 de la PC física. Tal como se expresó, esta interfaz servirá para la interconexión entre la red SDN virtualizada y la física (*legacy*). La red emulada en mininet será administrada por OVX/FV, quien tendrá a su vez la responsabilidad de virtualizar la red para que sea controlada por el controlador Floodlight.

Tanto la solución de virtualización como Floodlight se encuentran instalados en *dockers* independientes ambos corriendo en la PC física e interconectados con la misma a través de su interfaz *docker0*.

C. Objetivo

Comunicar el host emulado en mininet con dispositivos de la red legacy de la Figura 32 (Tablet e interfaz del router Cisco 7200). Notar que el host estará conectado con la red externa a través de una red virtualizada a través de OVX o FV, según corresponda. De esta manera se probará la capacidad del host para comunicarse fuera del dominio OF virtualizado.

El desarrollo de la simulación se presenta en ANEXO 17: Funcionamiento en redes híbridas.

ii. *Resultado*

Tanto las redes virtuales creadas a partir de OVX como FV son interoperables con las redes legacy configurando redes híbridas. Para los switches virtuales, cada *end-point* remoto conectado a la red física se ve como si estuviera conectado directamente en la interfaz del switch virtual. Esta situación genera una abstracción de la complejidad de la red *legacy* existente entre el dispositivo de red virtual y el *end-point* físico.

Si bien esta experiencia se realiza sobre una red sencilla, permite inferir que existe la posibilidad de integrar una red OF con una red legacy compleja de manera transparente.

15) Migración

Tal como se expresó en el apartado 3. ATRIBUTOS CARACTERÍSTICOS DE LAS ESTRATEGIAS DE VIRTUALIZACIÓN EN REDES DEFINIDAS POR SOFTWARE, el proceso de migración debe garantizar que los switches conserven el estado de sus tablas de flujo.

En la caracterización del atributo 9) Grabar el estado de configuración de una red (Snapshot) del presente capítulo, se expresó que tanto FV como OVX no cuentan con esta funcionalidad, es decir, no es posible grabar y en consecuencia restaurar el estado de configuración de los flujos en los switches. Esta restricción no permite el cumplimiento del presente atributo.

Adicionalmente en la documentación de OVX [63] se indica que la funcionalidad de "live migration" se encuentra en desarrollo.

16) Microsegmentación

Tanto FV como OVX permiten la segmentación de la red de manera granular, lo cual posibilita la generación del sustrato requerido para la microsegmentación de un centro de cómputos. Tal como se expresó en Aislación 2) Aislación del **Plano de datos** apartado ii. OpenVirtex, permite una mejor segmentación de la red ya que ofrece una ventaja en cuanto a la aislación del tráfico.

Sin embargo, ambas soluciones de virtualización no incluyen de manera nativa una funcionalidad que permita realizar el filtrado de tráfico este-oeste en el datacenter. A partir de ello, se puede afirmar que ninguna de las soluciones evaluadas cumple en forma completa con éste atributo.

C. COMPARACIÓN DE ESTRATEGIAS

Para generar una comparación objetiva de las estrategias abordadas se propone el desarrollo de un *framework.* A través de esta herramienta se busca obtener un puntaje para cada una de las soluciones de virtualización a evaluar, el cual será el resultado de la suma de los puntajes individuales correspondientes a cada atributo.

El puntaje total se define de la siguiente manera:

$Puntaje\ Total\ (PT) = (\sum_{x=1}^{n} PA_x)$ **(1)**; siendo PA_x el puntaje asociado a cada atributo y n el número total de atributos definidos.

El puntaje asociado a cada uno de los atributos PA_x se obtiene a partir del nivel de cumplimiento de dicho atributo y su peso relativo de acuerdo al contexto de aplicación de la solución de virtualización. De esta manera, el puntaje individual se define:

$$Puntaje\ Atributo\ (PA_x) = Nivel\ de\ cumplimiento\ (NC_x)\ x\ Peso\ relativo\ (PR_x) (\mathbf{2})$$

El nivel de cumplimiento NC_x expresa en que grado una solución de virtualización cumple con dicho atributo. NC_x tiene sentido *"local"* puesto que es independiente del contexto de aplicación.

Se obtiene de la siguiente manera:

$$Nivel\ de\ cumplimiento\ (NC_x) = \begin{cases} Factor\ de\ cumplimiento\ (FC_x)(Cumple) \\ 0\ (No\ cumple) \end{cases} \textbf{(3)}$$

El factor de cumplimiento representa un valor asociado al nivel de cumplimiento, que se utiliza para cuantificar el mismo.

Con el objetivo de darle sentido *"global"* al puntaje PA_x, cada atributo es ponderado en relación a su contexto de aplicación definiendo el peso relativo PR_x.

La comparación que se presenta a continuación se realiza tomando en consideración la aplicación en un centro de cómputo, tal como se presentó en b. Marco de trabajo. En consecuencia, los pesos relativos estarán contextualizados en dicho caso de uso. Independientemente de ello, *la idea es generalizable a otros escenarios* (enlaces WAN, redes de acceso, etc) *si se ajusta consecuentemente la ponderación de cada atributo.*

Teniendo en cuenta lo expuesto, es posible que uno o más atributos sean requeridos en forma excluyente. De darse este caso, la expresión asociada al puntaje total (PT), se altera de la siguiente manera:

$Puntaje\ Total\ (PT) = (\sum_{x=1}^{n} PA_x) x \frac{NC_a}{FC_a} x \frac{NC_b}{FC_b} x \dots x \frac{NC_N}{FC_N}$ (**4**) ; siendo PA_x el puntaje asociado a cada atributo, n el número total de atributos definidos y *a, b, … N* los atributos requeridos de manera excluyente.

De esta manera, si el atributo excluyente se cumple, la expresión original **(1)** se multiplica por uno manteniendo el puntaje, por el contrario, se multiplicaría por cero anulando la expresión. *Esta última condición expresa que la implementación de dicha solución resulta inviable* para el caso de uso bajo análisis. En consecuencia, se *propone* la evaluación preliminar de los atributos excluyentes, ya que en caso de no cumplimiento de alguno, *la solución analizada será descartada por completo, sin necesidad de evaluar los demás atributos*.

Si bien este trabajo compara y presenta resultados relativos a FV y OVX, la metodología propuesta es de aplicación general para el análisis de otras propuestas de virtualización en SDN.

1) Clasificación de niveles de cumplimiento asociados a los atributos definidos

i. *Aislación*

- Alta

La solución garantiza la aislación a nivel del plano de datos, control y esquemas de direccionamiento.

- Media

La solución no permite reservar recursos físicos (QoS). Sin embargo, garantiza la correcta aislación a nivel del plano de control, esquemas de direccionamiento y de los datos que circulan por cada red virtual.

- Baja

Garantiza aislación a nivel del plano de control y de datos, sin embargo, no permite la reserva de recursos físicos, ni tampoco superposición en los esquemas de direccionamiento.

ii. *Abstracción de la topología de red física*

- Alta

Permite la generación de abstracciones virtuales de múltiples switches y *links* físicos, y la implementación conjunta de ambas.

- Media

Permite la generación de abstracciones virtuales de múltiples switches y *links* físicos, pero no así la implementación en conjunto.

- Baja

Implementa solamente una de las abstracciones, links o switches.

iii. *Automatización en el despliegue*

- Flexible

Permite la generación de nuevas configuraciones de red a automatizar, a través de una herramienta que facilite el diseño y despliegue de las mismas.

- No flexible

El despliegue y diseño de las redes a automatizar, se realiza a través de un script y de manera manual.

iv. *Alta disponibilidad y resiliencia*

- Automatizada

Ante una falla de la red, la misma se reconfigura de manera automática para continuar la prestación de servicios.

- Manual

Ante una falla de la red, la misma dispara una alarma que permite que el administrador actúe reconfigurando la red en forma manual.

v. *Grabar el estado de configuración de una red*

- Completa

Permite grabar la configuración topológica, como así también el estado de las tablas de flujo en cada uno de los switches de la red (Snapshot).

- Parcial

Permite grabar la configuración topológica de la red, pero no el estado de los flujos asociados.

vi. *Virtualización recursiva*

- Híbrida

Permite delegar una red virtual a otra solución de virtualización, para que luego ésta virtualice nuevamente la red en forma recursiva.

- No híbrida

Permite la virtualización recursiva, pero solamente tiene compatibilidad con la misma solución (Ej. FV con FV u OVX con OVX).

vii. *Degradación de performance*

Este atributo cuantifica el OH asociado a la implementación de una solución de virtualización. El mismo se manifiesta en *delay* y se expresa en ms.

En este caso, el nivel de cumplimiento no se puede clasificar en forma discreta, sino que se deben medir los niveles de retardo generados y contrastarlos con los parámetros de performance requeridos dependiendo de la aplicación [65].

Nota: las soluciones de virtualización estudiadas generan un *delay* que no es significativo, tal como se expresó en b.Caracterización de atributos en base a los resultados de la simulación. En la generalidad de los casos, estos niveles de delay no deberían generar inconvenientes, aunque se deben tomar en consideración al momento de diseñar una red. A los fines de la comparación, se define que ambas soluciones cumplen con este atributo.

viii. *Microsegmentación*

- Seguridad

La solución de virtualización tiene la capacidad de segmentar el tráfico de la red en forma granular e implementa funcionalidades de seguridad (*firewall* distribuido, filtrado de tráfico, etc).

- Networking

La solución de virtualización tiene la capacidad de segmentar el tráfico de la red de forma granular, pero no implementa ninguna característica de seguridad asociada.

Para todos los casos, los niveles de cumplimiento se cuantifican en forma normalizada.

Para los atributos no mencionados en el presente apartado, no se definen niveles; su análisis se simplifica al cumplimiento o no de los mismos. Dicha definición se deja para futuros trabajos.

2) Peso relativo de los atributos definidos en el contexto de un centro de cómputos

Tal como se expuso, la definición de pesos globales depende de la importancia de un determinado atributo, de acuerdo al caso de uso abordado. Sin embargo, *esta definición no es de carácter absoluto, y es donde el usuario de la metodología propuesta puede caracterizar la importancia relativa de cada atributo, adaptándola a su caso de uso específico.*

Dicho de otra manera, no todos los centros de cómputo persiguen la implementación de las mismas funcionalidades con el mismo grado de importancia o profundidad. Es así que, por ejemplo, el centro de cómputos de una determinada compañía podría tener como principal objetivo la agilidad de cara a sus clientes, lo cual otorgaría el mayor peso a los atributos relacionados (automatización en el despliegue, políticas extensibles, etc.), mientras que otra podría optar por potenciar las cualidades *multitenancy* lo cual requeriría, por ejemplo, de un alto nivel de aislación.

La metodología propuesta se utilizará para comparar las estrategias de virtualización bajo estudio. En este contexto, y en base al escenario planteado en b. Marco de trabajo, se propondrán pesos relativos a cada atributo para la caracterización de las soluciones de virtualización en el marco de uso de un *datacenter* corporativo.

Dichos pesos se definen en base a artículos publicados por diversas consultoras [66] [67] [16] en relación al *roadmap* para los centros de cómputo y tomando como referencia los requerimientos de una empresa de la industria de *oil & gas* para su *datacenter*.

En función de ello, se ponderan los atributos definidos en relación a su aporte para el cumplimiento de los siguientes aspectos:

i. *Aislación de los datos, resiliencia, performance y funcionamiento en redes híbridas.*
ii. *Agilidad en el aprovisionamiento de nuevos servicios*
iii. *Integración con la nube (Datacenters Híbridos)*
iv. *Multitenancy*

Los atributos serán ponderados en este caso en cuatro niveles; los de mayor peso serán los relacionados con funcionalidades de base (aislación de los datos, resiliencia, *performance* e integración con redes tradicionales), mientras que los de menor peso son los que aportan facilidades de *multitenancy*. En general, esta última característica es deseable para *datacenters* de *carriers* que proveen servicios en la nube y en consecuencia deben soportar múltiples grupos de usuarios e integración con otros centros de cómputo. En contraposición, los primeros resultan en características fundamentales para cualquier tipo de *datacenter* y por ello se le atribuye el mayor peso. Además, se los considera excluyentes para viabilizar su implementación.

Este criterio se adopta para la presente comparación, sin efecto de ello, la ponderación de los atributos se deja librada a la definición por parte del usuario de la metodología, realizando un correcto análisis de sus requerimientos.

La comparación de ambas estrategias en base a los niveles de cumplimiento definidos y la ponderación de cada atributo se presenta en la Tabla 2.

Tabla 2 -Comparación de FV y OVX en el contexto de un centro de cómputos corporativo

Comparación de FV y OVX en el contexto de un centro de cómputos corporativo												Puntaje Atributo	
ID	Atributo	Excluyente	Clasificación		FV	OVX	Factor de cumplimiento		FV	OVX	Peso Relativo	Peso Normalizado	FV	OVX
1	Transparencia		Cumple	1	•	•					4	9,09	9,09	9,09
			No Cumple	0										
2	Aislación	x	Cumple	1	•	•	Alto	1,00			4	9,09	3,00	6,00
							Medio	0,66		•				
							Bajo	0,33	•					
			No Cumple	0										
3	Administración centralizada de políticas		Cumple	1	•	•					3	6,82	6,82	6,82
			No Cumple	0										
4	Políticas extensibles		Cumple	1	•*	•					3	6,82	6,82	6,82
			No Cumple	0										
5	Reconfiguración en caliente		Cumple	1	•*	•					3	6,82	6,82	6,82
			No Cumple	0										
6	Abstracción de la topología de red física		Cumple	1		•	Alto	1,00		•	3	6,82	0,00	6,82
							Medio	0,66						
							Bajo	0,33						
			No Cumple	0	•									
7	Automatización en el despliegue		Cumple	1		•	Flexible	1,00			3	6,82	0,00	3,41
							No Flexible	0,50		•				
			No Cumple	0	•									

8	Alta disponibilidad y resiliencia	x	Cumple	1		•	Automatizada	1,00		•	4	9,09	0,00	9,09
							Manual	0,50						
			No Cumple	0	•									
9	Grabar el estado de configuración de una red		Cumple	1	•	•	Completa	1,00			3	6,82	3,41	3,41
							Parcial	0,50	•	•				
			No Cumple	0										
10	Multitenancy		Cumple	1	•	•					1	2,27	1,00	1,00
			No Cumple	0										
11	Virtualización recursiva		Cumple	1	•	•	Hibrida	1,00	•	•	1	2,27	2,27	2,27
							No Hibrida	0,50						
			No Cumple	0										
12	Degradación de performance (Overhead)	x	Valor numérico	#	0.6 ms	0.3 ms		1,00	•	•	4	9,09	9,09	9,09
13	Retrocompatibilidad con redes legacy		Cumple	1							1	2,27	0,00	0,00
			No Cumple	0	•	•								
14	Funcionamiento en redes hibridas	x	Cumple	1	•	•					4	9,09	9,09	9,09
			No Cumple	0										
15	Live migration		Cumple	1							2	4,55	0,00	0,00
			No Cumple	0	•	•								
16	Microsegmentación		Cumple	1	•	•	Seguridad	1,00			1	2,27	1,14	1,14
							Networking	0,50	•	•				
			No Cumple	0										
											Puntaje parcial **		58,5	80,9
											Puntaje total ***		0,0	80,9

*Por diseño se contempla la funcionalidad, sin embargo, en ocasiones las actualizaciones no se vieron reflejadas a nivel del controlador. Esta situación se dio en forma aleatoria, no pudiendo determinar las causas que lo originan.

**El puntaje parcial no contempla la calidad de excluyentes de los atributos señalados

*** El puntaje total considera la calidad de excluyentes de los atributos señalados. De acuerdo a la expresión (4) y debido a que FV no cumple con el atributo de alta disponibilidad y resiliencia, se concluye en que no es viable su implementación en el caso de uso analizado.

Nota: los valores relativos a degradación de performance (12), se obtuvieron de “OpenVirteX: Make Your Virtual SDNs Programmable” [43]

3) Análisis de los resultados

Surge de la aplicación de la metodología de comparación propuesta que OVX es superior a FV.

Las principales diferencias se presentan en los siguientes atributos:

Aislación: OVX y FV presentan las mismas limitaciones al respecto de la asignación de recursos físicos a cada una de las redes virtuales. Dicha limitación tiene en común la utilización de OF 1.0 por parte de ambas.

Sin embargo, OVX se diferencia en que tiene la capacidad de utilizar la totalidad del espacio de flujos, lo cual permite la superposición en la definición de entradas de flujo y esquemas de direccionamiento asociados a diferentes redes virtuales.

Políticas extensibles y reconfiguración en caliente: el diseño de FV contempla la funcionalidad, sin embargo, en ocasiones las reconfiguraciones no se vieron reflejadas a nivel del controlador. Esta situación se dio en forma aleatoria, no pudiendo determinar las causas que lo originan.

Abstracción de la topología de red física: FV no permite la generación de redes virtuales en forma independiente de la topología física, sino que su definición está supeditada al diseño de la misma. En contraposición, OVX implementa switches virtuales y links virtuales que permiten la abstracción de la topología de red subyacente.

Automatización en el despliegue: OVX permite la implementación de redes de manera automatizada a través de scripts y un componente (“embedder”) diseñado para tal fin. FV no implementa ningún mecanismo de automatización.

Alta disponibilidad y resiliencia: OVX implementa mecanismos de resiliencia ante la falla de un enlace, permitiendo la reestructuración de la red a través de caminos alternativos. En contraposición FV no dispone de ningún mecanismo similar.

Degradación de performance: Tanto OVX como FV generan delay debido a la manipulación del tráfico entre los controladores y los dispositivos de red.

Si bien el retardo generado por la implementación de OVX es menor comparado con FV, en ambos casos no es significativo. Debido a ello, se considera que ambas soluciones cumplen con éste atributo.

El no cumplimiento del atributo de alta disponibilidad y resiliencia, hace *inviable la aplicación de FV para el caso de uso definido* tal como se aprecia en la Tabla 2.

Además, ambas soluciones utilizan como mecanismo de virtualización la manipulación de flujos entre el controlador y los switches, siendo FV el precursor de este mecanismo. Resulta razonable apreciar una evolución por parte de OVX en este sentido.

IV.CONCLUSIÓN

1.CONCLUSIONES

Se presentó el potencial de SDN en la virtualización de las redes de los centros de cómputo. Si bien SDN es transversal a toda la red, se centró el estudio sobre el contexto de un *datacenter* por ser el caso de uso más significativo. En este contexto, se definieron los atributos requeridos por parte de una solución de virtualización y una metodología que permite su caracterización y comparación. En particular, se realizaron simulaciones que permitieron la implementación de dos soluciones de virtualización OVX y FV.

Se evaluaron los atributos característicos de las soluciones de virtualización en SDN en general, y de dos soluciones específicas en particular. Se realizó una comparación que puso de manifiesto las ventajas de OVX frente a FV para el caso de uso abordado, permitiendo explorar la evolución en los hipervisores centralizados basados en la manipulación de flujos de control.

Surge de la evaluación, también, que existen funcionalidades deseadas que no son soportadas por ninguna de las dos soluciones – como, por ejemplo, live migration que resulta fundamental para la integración con ambientes *cloud* (datacenters híbridos) -, lo cual sugiere que aún se requieren esfuerzos para viabilizar su implementación en ambientes productivos.

La definición de atributos y la metodología de caracterización desarrollada no están acotadas a las implementaciones abordadas, sino que tienen por objetivo servir de framework para la futura evaluación en otras configuraciones. En este sentido, se sugieren futuras líneas de investigación en torno a la caracterización de ACI (Cisco), NSX (VMware) y xDPd (eXtensible OpenFlow DataPath daemon), que están tomando relevancia en el mercado.

Los resultados específicos obtenidos a través de las simulaciones y el relevamiento bibliográfico realizado permiten confirmar algunas de las ventajas de SDN con respecto al paradigma tradicional. Entre ellas se destacan: la agilidad de los despliegues a través de la automatización; la capacidad de gestión centralizada, facilitando la administración; la posibilidad de desarrollo continuo de funcionalidades sin cambiar el HW; la baja complejidad para la creación de ambientes multitenancy; la provisión de IaaS y los procesos de migración. Además, la visión centralizada de la red simplifica su operación

e implica no utilizar protocolos de control que permiten evitar bucles e implementar políticas de *forwarding* (ruteo y STP), los cuales, trabajan en forma independiente, complejizan la red y dificultan el *troubleshooting* en caso de falla.

Del análisis bibliográfico y de mercado, surge además que los referentes más importantes del *networking* global están invirtiendo en desarrollos en torno a esta tecnología. Lo mismo ocurre desde la academia. Las Universidades más prestigiosas están realizando investigaciones o desarrollos en la materia. Si bien aún no está claro el *roadmap* y, por tanto, donde va converger esta tecnología - sea por la falta de estandarización existente hasta el momento o por la puja de los incumbentes y los ámbitos académicos para imponer sus propias visiones -, existe un horizonte claro de evolución hacia la programabilidad de las redes.

2.APORTE DEL TRABAJO PROPUESTO

El estudio de técnicas y mecanismos que permitan la virtualización de las redes es una temática de actualidad. En particular resulta de interés el estudio, análisis e investigación de diferentes alternativas que surgen a partir de SDN. Esta se presenta en “Software-Defined Networking: A Comprehensive Survey” [10] como una de las áreas de interés sobre las cuales aún se requieren esfuerzos de investigación y desarrollo.

En este sentido aporta valor agregado la definición de atributos que permiten su caracterización y posibilitan realizar una comparación objetiva utilizando la metodología presentada, dirigida en particular para aquellos que tienen en su *roadmap* la implementación de SDN con foco en la virtualización. Al respecto, existen publicaciones que arrojan sólo resultados de una u otra técnica de virtualización de la red o se acotan a la comparación sobre algún atributo específico. El presente trabajo propone un abordaje holístico, entendiendo las redes como un todo, avanzando sobre un análisis global de las variables características.

Por otro lado, las simulaciones realizadas, la caracterización y comparación de las estrategias abordadas, aportan nuevas experiencias e información que se complementa o suma a la existente respecto de estas tecnologías.

3.PRÓXIMOS PASOS

Se proponen lineamientos de investigación que pudieran aportar valor dando continuidad al presente trabajo o que no fueran abordados y resulten interesantes para el estado del arte.

En lo que respecta a la metodología presentada se propone:

- Evaluación de otras tecnologías de virtualización en SDN como ser ACI (Cisco), NSX (VMware), xDPd (OpenFlow eXtensible DataPath daemon).
- Identificar niveles de cumplimiento para aquellos atributos en los cuales no fueron definidos.

En lo que respecta a las simulaciones realizadas:

- Replicar el diseño y ejecución sobre redes reales.
- Realizar pruebas integrando aplicaciones SDN a través de la interfaz norte de los controladores y NFV (Network Function Virtualization).
- Probar la integración de SDN con redes legacy en configuraciones más complejas.
- Realizar pruebas específicas de robustez de los mecanismos de aislación en el plano de control, datos y esquemas de direccionamiento.

En lo que respecta a diseño y funcionalidades de las soluciones de virtualización abordadas:

- Implementar la compatibilidad con OF 1.3 y superiores.
- Implementar funcionalidades de “live migration”.
- Realizar pruebas de escalabilidad en lo referente a virtualizar la red recursivamente.
- Implementar la funcionalidad para grabar el estado de los flujos en los switches.

V.BIBLIOGRAFÍA

[1] D. D. Clark, C. Communication, and R. Vol, “The Design Philosophy of the DARPA Internet Protocols 2 . Fundamental Goal,” pp. 102–111, 1988.

[2] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network,” *SIGCOMM, A ugust 17-21*, pp. 183–197, 2015.

[3] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, “Software-Defined Networking: Challenges and research opportunities for Future Internet,” *Comput. Networks*, vol. 75, pp. 453–471, 2014.

[4] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks [white paper],” *ONF White Pap.*, pp. 1–12, 2012.

[5] N. Mckeown, T. Anderson, L. Peterson, J. Rexford, S. Shenker, and S. Louis, “OpenFlow : Enabling Innovation in Campus Networks,” 2008.

[6] J. Postel, “RFC 791: Internet Protocol,” *Ietf Rfc 791*. p. 10, 1981.

[7] T. (University of W. Benson, A. (University of W. Akella, and D. (Microsoft R. Maltz, “Unraveling the complexity of network management,” *Nsdi ’09*, pp. 335–348, 2009.

[8] A. Greenberg, G. Hjalmtysson, D. a. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, “A clean slate 4D approach to network control and management,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, p. 41, 2005.

[9] “Open Networking Lab.” [Online]. Available: https://www.opennetworking.org/.

[10] D. Kreutz, F. M. V Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, D. Kreutz, and F. Ramos, “Software-Defined Networking: A Comprehensive Survey,” *VERSION 4.0 — TO Appear Proc. IEEE*, vol. 103, no. 1.

[11] SDxCentral, “SDN Controllers Report 2015,” *SDxCentral*, 2015.

[12] Open Networking Foundation, “OpenFlow Switch Specification,” 2013.

[13] Open Networking Foundation, “OpenFlow Switch Specification,” *ONF TS-001*, vol. 0, pp. 1–36, 2009.

[14] “HPE SDN App Store.” [Online]. Available: https://saas.hpe.com/marketplace/sdn.

[15] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” *Building*, vol. 54, p. 162, 2000.

[16] “SDxCentral.” [Online]. Available: www.sdxcentral.com.

[17] VMware, “Virtualization overview,” *White Pap. http//www. vmware. com/pdf/virtualization. pdf*, pp. 1–11, 2006.

[18] A. Virtualization, “Oracle VM 3 :,” no. August 2011, pp. 1–287, 2011.

[19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 164, 2003.

[20] A. Kivity, U. Lublin, A. Liguori, Y. Kamay, and D. Laor, “kvm: the Linux virtual machine monitor,” *Proc. Linux Symp.*, vol. 1, pp. 225–230, 2007.

[21] “Oxford Dictionaries.” [Online]. Available: http://www.oxforddictionaries.com.

[22] a. Alba, G. Alatorre, C. Bolik, a. Corrao, T. Clark, S. Gopisetty, R. Haas, R. I. Kat, B. S. Langston, N. S. Mandagere, D. Noll, S. Padbidri, R. Routray, Y. Song, C. Tan, and a. Traeger, “Efficient and agile storage management in software defined environments,” *IBM J. Res. Dev.*, vol. 58, no. 2, pp. 1–12, 2014.

[23] C.-S. Li, B. L. Brech, S. Crowder, D. M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, S. Pappe, B. Rajaraman, J. Rao, R. P. Ratnaparkhi, R. a. Smith, and M. D. Williams, “Software defined environments: An introduction,” *IBM J. Res. Dev.*, vol. 58, no. 2, pp. 1–11, 2014.

[24] T. Sukigara and N. Kumagai, “Hitachi Storage Solution for Cloud Computing,” vol. 61, no. 2, pp. 85–89, 2012.

[25] H. D. Systems, “Reduce Costs and Risks for Data Migrations,” no. April, 2015.

[26] M. B. K. Nance, B. Hay, “Virtual Machine Introspection,” *Secur. Privacy, IEEE*, vol. 6, no. 5, pp. 32–37, 2008.

[27] “VMware.” [Online]. Available: http://www.vmware.com/.

[28] “Microsoft HYPER-V Virtualization.” [Online]. Available: https://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx.

[29] VMware, “Understanding Full Virtualization, Paravirtualization, and Hardware Assist,” *Memory*, p. 17, 2007.

[30] Oracle, “How to Consolidate Servers and Applications with Oracle ® Solaris Containers,” 2011.

[31] “Docker.” [Online]. Available: www.docker.com.

[32] “Docker Hub.” [Online]. Available: https://hub.docker.com/.

[33] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, “FlowVisor : A Network Virtualization Layer FlowVisor : A Network Virtualization Layer,” no. December 2015, 2009.

[34] C. V. Design, “Network Virtualization — Path Isolation Design,” *Design*, 2009.

[35] M. Casado, N. Foster, and a Guha, “Abstractions for software-defined networks,” *Commun. ACM*, 2014.

[36] C. Peng, M. Kim, Z. Zhang, and H. Lei, “VDN: Virtual machine image distribution network for cloud data centers,” *Proc. - IEEE INFOCOM*, pp. 181–189, 2012.

[37] M. Mahalingam, D. G. Hutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks,” no. October, pp. 1–23, 2014.

[38] R. Doriguzzi-Corin, E. Salvadori, M. Gerola, M. Sune, and H. Woesner, “A Datapath-Centric Virtualization Mechanism for OpenFlow Networks,” *2014 Third Eur. Work. Softw. Defin. Networks*, pp. 19–24, 2014.

[39] D. Depaoli, R. Doriguzzi Corin, M. Gerola, and E. Salvadori, “Demonstrating a Distributed and Version-agnostic OpenFlow Slicing Mechanism,” *Ewsdn*, 2014.

[40] R. D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, “VeRTIGO: Network virtualization and beyond,” *Proc. - Eur. Work. Softw. Defin. Networks, EWSDN 2012*, pp. 24–29, 2012.

[41] K. Rausch, “Network Virtualization - An Overview,” *Semin. FI IITM SS 2011, Netw. Archit. Serv. July 2011 153*, no. July, 2011.

[42] A. Al-Shabibi and M. De Leenheer, “OpenVirteX: A Network Hypervisor,” *Open*

Netw. Summit, 2014.

[43] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, “OpenVirteX: Make Your Virtual SDNs Programmable,” *Proc. Third Work. Hot Top. Softw. Defin. Netw.*, pp. 25–30, 2014.

[44] SDxCentral, “SDxCentral Network Virtualization Report,” 2015.

[45] “FlowVisor.” [Online]. Available: https://github.com/OPENNETWORKINGLAB/flowvisor/wiki.

[46] “OpenVirtex.” [Online]. Available: http://ovx.onlab.us/.

[47] T.-Y. H. Rob Sherwood, Michael Chan, Glen Gibb, Nikhil Handigol and and N. M. Peyman Kazemian, Masayoshi Kobayashi, David Underhill, Kok-Kiong Yap, “Carving research slices out of your production networks with OpenFlow,” 2009.

[48] “OpenVirtex.” [Online]. Available: http://ovx.onlab.us/documentation/api/.

[49] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, “Survey on Network Virtualization Hypervisors for Software Defined Networking,” *going Pap.*, no. c, pp. 1–27, 2015.

[50] P. Skoldstrom and K. Yedavalli, “Network virtualization and resource allocation in OpenFlow-based wide area networks,” *2012 IEEE Int. Conf. Commun.*, pp. 6622–6626, 2012.

[51] N. Kang, Z. Liu, J. Rexford, and D. Walker, “Optimizing the ‘One Big Switch’ Abstraction in Software-Defined Networks,” *Conext’13*, p. 17, 2013.

[52] A. Menon, J. R. Santos, Y. Turner, G. (John) Janakiraman, and W. Zwaenepoel, “Diagnosing Performance Overheads in the Xen Virtual Machine Environment,” *Proc. 1st ACM/USENIX Int. Conf. Virtual Exec. Environ. (VEE 2005)*, no. June, pp. 13–23, 2005.

[53] J. P. Walters, V. Chaudhary, C. Minsuk, S. Guercio, and S. Gallo, “A comparison of virtualization technologies for HPC,” *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA*, pp. 861–868, 2008.

[54] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, “OSHI -Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds).”

[55] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, J. Rexford, and D. Walker, “Transparent, Live Migration of a Software-Defined Network,” *Proc. ACM Symp. Cloud Comput.*, pp. 3:1–3:14, 2014.

[56] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, “Live migration of an entire network (and its hosts),” *Proc. 11th ACM Work. Hot Top. Networks - HotNets-XI*, pp. 109–114, 2012.

[57] T. Jackson, “NSX Micro-segmentation,” 2014.

[58] vmware, “Data Center Micro - Segmentation,” no. January, pp. 1–24, 2014.

[59] “Mininet.” [Online]. Available: http://mininet.org/.

[60] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using Mininet for emulation and prototyping Software-Defined Networks,” *2014 IEEE Colomb. Conf. Commun. Comput. COLCOM 2014 - Conf. Proc.*, no. May 2016, 2014.

[61] “FlowVisor.” [Online]. Available: https://github.com/opennetworkinglab/flowvisor/wiki/API-Changes#Monitoring_API.

[62] “JSON-RPC.” [Online]. Available: http://www.jsonrpc.org/specification.

[63] “OpenVirtex.” [Online]. Available: http://ovx.onlab.us/documentation/architecture/operation-and-subsystems/.

[64] “MongoDB.” [Online]. Available: https://www.mongodb.com/.

[65] Y. Chen, T. Farley, and N. Ye, “QoS Requirements of Network Applications on the Internet,” *Information-Knowledge-Systems Manag.*, vol. 4, no. 1, pp. 55–76, 2004.

[66] “Gartner.” [Online]. Available: http://www.gartner.com/.

[67] “Forrester.” [Online]. Available: https://go.forrester.com/.

[68] “Floodlight OpenFlow Controller.” [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[69] “The Apache Software Foundation.” [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0.

[70] “Big Switch.” [Online]. Available: http://www.bigswitch.com/.

[71] “OpenDayLight.” [Online]. Available: https://www.opendaylight.org/.

[72] “Wireshark.” [Online]. Available: https://www.wireshark.org/.

[73] “iPerf.” [Online]. Available: https://iperf.fr/.

[74] “Mininet host talking to Internet.” [Online]. Available: https://techandtrains.com/2013/11/24/mininet-host-talking-to-internet/.

[75] “GNS3 Forum.” [Online]. Available: http://forum.gns3.net/topic18.html.

VI.ANEXOS

1.ANEXO 1: HERRAMIENTAS UTILIZADAS EN LA SIMULACIÓN

A continuación, se listan las herramientas utilizadas para la simulación:

i. *Floodlight* [68]

Floodlight es un controlador de clase *enterprise open source*, bajo licencia Apache [69]. El mismo se encuentra desarrollado en Java y está soportado por una comunidad abierta de desarrollo e ingenieros de Big Switch Networks [70].

ii. *OpenDayLight* [71]

ODL es un proyecto colaborativo *open source* bajo Linux Foundation, cuyo objetivo, es la adopción de SDN y la creación de los cimientos que posibiliten la implementación de NFV.

ODL está compuesta por una comunidad de desarrollo y miembros representantes de los mayores fabricantes de equipamiento de networking. Entre los fundadores se encuentran: Arista Networks, Big Switch Networks, Brocade, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Nuage Networks, PLUMGrid, Red Hat, and VMware. (Nota: Juniper, Microsoft, and VMware se retiraron en 2015) [16].

ODL ofrece una plataforma completa de SDN, sin embargo, en el presente trabajo se utiliza únicamente como controlador.

iii. *Docker* [31]

Utiliza tecnología de containers para virtualizar una instancia de OS con aplicaciones pre instaladas y sus dependencias, permitiendo “empaquetarlo y desplegarlo” en cualquier otro OS de manera rápida. A su vez, la solución cuenta con un sitio llamado “Docker Hub”[32] el cual sirve de repositorio para el almacenamiento y descarga de estos

contenedores, permitiendo el despliegue de nuevos entornos virtuales en cuestión de minutos y gran flexibilidad ante requerimientos de portabilidad de los mismos.

iv. *Wireshark* [72]

Wireshark es un analizador de protocolos muy utilizado en la industria y en las Universidades.

Permite el análisis y filtrado de tráfico, lo cual lo convierte en una solución muy potente para la resolución de problemas en las redes de datos. También representa una herramienta muy didáctica para el análisis y enseñanza de protocolos de comunicaciones.

v. *iPerf* [73]

iPerf es una herramienta que permite la medición de BW en redes IP. Para cada prueba, genera un reporte del BW, pérdidas, entre otros parámetros.

2.ANEXO 2: SCRIPT DE LA RED EMULADA EN MININET

```
#!/usr/bin/python


from mininet.topo import Topo


class RedTopo(Topo):


  def __init__(self):

    # Initialize topology
```

```
        Topo.__init__(self)


        # Create template host, switch, and link
        hconfig = {'inNamespace':True}
        GB_link_config = {'bw': 1}
        GGB_link_config = {'bw': 10}
        host_link_config = {}


        # Create switch nodes
        for i in range(6):
            sconfig = {'dpid': "%016x" % (i+1)}
            self.addSwitch('s%d' % (i+1), **sconfig)


        # Create host nodes
        for i in range(6):
            self.addHost('h%d' % (i+1), **hconfig)


        # Add switch links
    self.addLink('s1', 's2', **GGB_link_config)
    self.addLink('s1', 's2', **GGB_link_config)
    self.addLink('s3', 's1', **GGB_link_config)
```

```
        self.addLink('s3', 's2', **GGB_link_config)
        self.addLink('s4', 's1', **GGB_link_config)
        self.addLink('s4', 's2', **GGB_link_config)
        self.addLink('s5', 's1', **GGB_link_config)
        self.addLink('s5', 's2', **GGB_link_config)
        self.addLink('s6', 's1', **GGB_link_config)
        self.addLink('s6', 's2', **GGB_link_config)


        # Add host links
        self.addLink('h1', 's3', **GGB_link_config)
        self.addLink('h1', 's4', **GGB_link_config)
        self.addLink('h2', 's3', **GB_link_config)
        self.addLink('h2', 's4', **GB_link_config)
        self.addLink('h3', 's5', **GGB_link_config)
        self.addLink('h3', 's6', **GGB_link_config)
        self.addLink('h4', 's5', **GB_link_config)
        self.addLink('h4', 's6', **GB_link_config)
        self.addLink('h5', 's1', **GGB_link_config)
        self.addLink('h6', 's2', **GB_link_config)


topos = { 'redtopo': ( lambda: RedTopo() ) }
```

3.ANEXO 3: CONFIGURACIÓN FLOWVISOR

Para iniciar el docker correspondiente a FlowVisor, desde el terminal del SO se debe ejecutar el siguiente comando:

```
# sudo docker run -it mno808/flowvisor:v2.0 bash
```

Luego para iniciar FlowVisor se debe ejecutar:

```
# flowvisor -u flowvisor flowvisor
```

Otra opción es la siguiente:

```
# sudo /etc/init.d/flowvisor start
```

Luego para configurar FlowVisor, es necesario abrir una instancia del docker en ejecución:

```
# sudo docker ps
# sudo docker exec -it <docker_id> bash
```

La configuración básica de FV implica la creación de un slice o red virtual y los espacios de flujo o flowspaces para cada una de ellas. Se expone la configuración genérica a continuación:

Creación de una red virtual:

```
# fvctl add-slice [opciones] <nombre_del_slice>
tcp:<ip_del_controlador>:<puerto> <email_del_administrador>
```

Configuración del espacio de flujo asociado a una red virtual:

```
# fvctl add-flowspace [opciones] <nombre_del_espacio_de_flujo>
<dpid> <prioridad> <match> <slice-perm>
```

Parámetros:

dpid: ip del switch a configurar. Éste parámetro depende de la red subyacente (en este caso la simulada a través de mininet).

priority: cuando el encabezado de un paquete es coincidente con múltiples espacios de flujo. Flowvisor lo asigna al de mayor prioridad.

match: describe los flujos asociados a la red virtual. Para ello se utilizan los campos permitidos en OpenFlow 1.0 (Figura 8 - Campos de las cabeceras de los paquetes usados para comparar contra las entradas de flujo en OF 1.0 [13]) separados por comas.

slice-perm: lista de redes virtuales que tienen control sobre el espacio de flujo.

4.ANEXO 4: CONFIGURACIÓN OPENVIRTEX

Para iniciar el docker correspondiente a OpenVirtex, desde el terminal del SO se debe ejecutar el siguiente comando:

```
# sudo docker run -it mno808/openvirtex:v3 bash
```

Luego para iniciar OpenVirtex se debe ejecutar desde el directorio de instalación:

```
# cd scripts
# sh ovx.sh
```

Luego para configurar OVX, es necesario abrir una instancia del docker en ejecución:

```
# sudo docker ps
# sudo docker exec -it <docker_id> bash
```

La configuración de redes virtuales se realiza a través de la API correspondiente. Para ello desde el directorio de instalación de OpenVirtex:

```
# cd utils
// la API se accede a través del script ovxctl.py
```

La configuración básica de OpenVirtex implica:

- Creación de una red virtual
- Creación de switches virtuales
- Creación de puertos virtuales
- Creación de links virtuales
- Conexión de hosts

Se expone la configuración genérica a continuación:

Creación de una red virtual:

```
# python ovxctl.py -n createNetwork
tcp:<ip_del_controlador>:<puerto_del controlador>
<direccion_de_red_de_los_host> <cant_de_host>


Ej.
# python ovxctl.py -n createNetwork tcp:172.17.0.2:10000 192.168.1.0 8
```

Creación de switches virtuales:

```
# python ovxctl.py -n createSwitch <tenant_id>
<dpid_del_switch_fisico>
```

Parámetros:

tenant_id: identificador de la red virtual

dpid_del_switch_fisico: id del switch físico a virtualizar. Para la presente simulación, el dpid dependerá de la configuración que se le imponga a cada switch a través del script de mininet.

Creación de puertos virtuales:

```
# python ovxctl.py -n createPort <tenant_id> <dpid_del_switch_fisico>
<puerto_fisico>
```

Parámetros:

puerto físico: número de puerto correspondiente al switch físico a virtualizar.

Creación de links virtuales:

```
# python ovxctl.py -n connectLink <tenant_id>
<dpid_del_switch_virtual_origen> <puerto_virtual_origen>
<dpid_del_switch_virtual_destino> <puerto_virtual_destino>
<modo_de_ruteo> <numero_de_rutas_de_bkup>
```

Parámetros:

dpid_del_switch_virtual_origen/destino: id del switch virtual origen/destino. El mismo se obtiene como output de la creación de un switch virtual.

puerto_virtual_origen/destino: id del puerto virtual origen/destino. El mismo se obtiene como output de la creación de un puerto virtual.

modo_de_ruteo: admite dos modos spf y manual. spf significa *shortest path first* lo cual implica que la conexión entre los dos switches virtuales se dará a través del camino más corto entre ellos. La otra alternativa es modo manual, en cuyo caso se indica el camino utilizando el parámetro setLinkPath disponible en la API de configuración.

numero_de_rutas_de_bkup: en caso de falla de la ruta principal, éste parámetro indica el número de caminos alternativos.

Conexión de hosts:

```
# python ovxctl.py -n connectHost <tenant_id>
<dpid_del_switch_virtual> <puerto_virtual> <MAC_address_host>
```

Luego para iniciar la red virtual:

```
# python ovxctl.py -n startNetwork <tenant_id>
```

5.ANEXO 5: CONFIGURACIÓN OPENDAYLIGHT

Para iniciar el docker correspondiente a ODL, desde el terminal del SO se debe ejecutar el siguiente comando:

```
# sudo docker run -it mno808/opendaylight:v3 bash
```

El controlador se inicia ingresando al directorio de instalación y ejecutando el siguiente comando:

```
# ./bin/karaf.
```

Para el correcto funcionamiento del controlador, se requieren algunos componentes no instalados por defecto. Para su instalación, se debe ejecutar el siguiente comando:

```
feature:install odl-base-all odl-aaa-authn odl-restconf odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs odl-l2switch-switch odl-dlux-core
```

OpenDayLight tiene un servicio web el cual permite entre otras funcionalidades, la visualización de la topología de red. Para acceder a dicho servicio, se debe ingresar en el navegador:

```
http://<ip_del_controlador>:8181/index.html
```

Luego del proceso de autenticación, se podrá visualizar la topología de red creada Figura Anexo 1.

- Nota: se debe ejecutar el comando pingall desde mininet para que el controlador visualice la topología de red subyacente. A partir de ello, se cargarán las tablas de flujo de los switches.

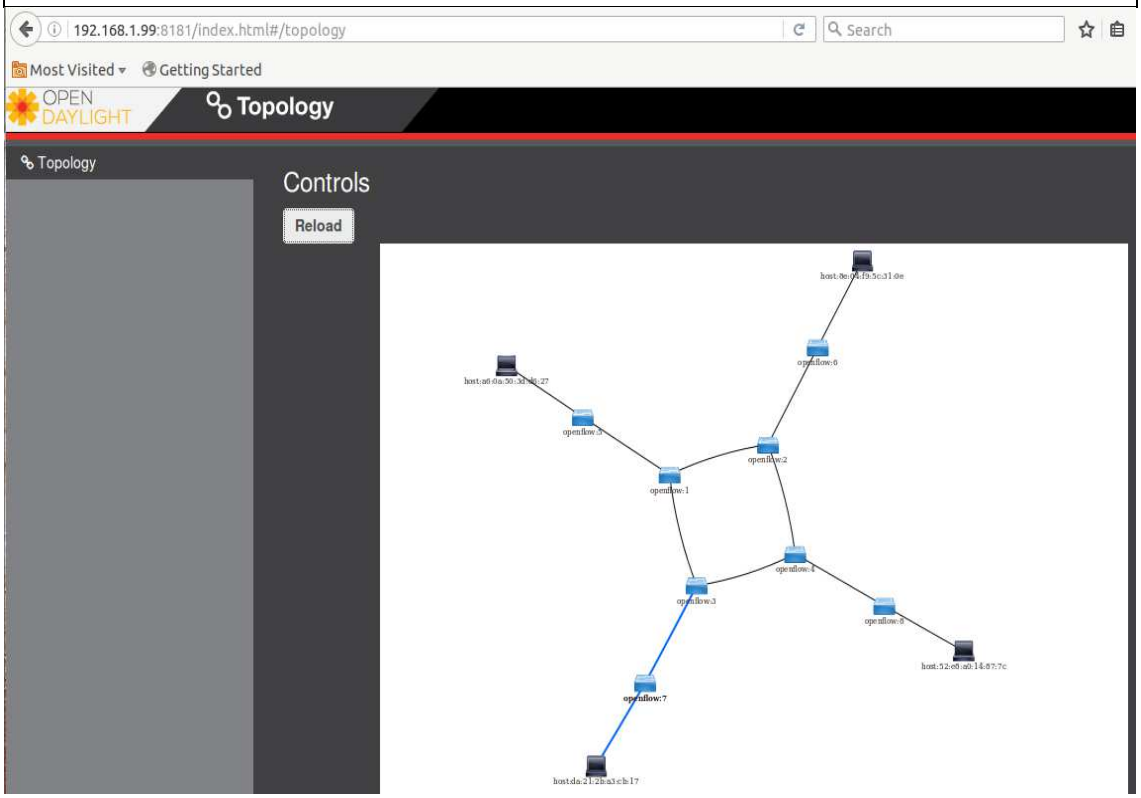


Figura Anexo 1 - Vista de topología de red ejemplo mediante ODL

6.ANEXO 6: CONFIGURACIÓN FLOODLIGHT

Para iniciar el docker correspondiente a Floodlight, desde el terminal del SO se debe ejecutar el siguiente comando:

```
# sudo docker run -it mno808/floodlight:v0 bash
```

En forma previa al inicio del controlador, se debe modificar la siguiente línea de su archivo de configuración, para permitir que el controlador se comunique con la capa de virtualización utilizando OF 1.0. Para ello, desde el directorio de instalación ejecutar:

```
# cd src/main/resources/
# nano floodlightdefault.properties
//modificar la siguiente línea dejando únicamente 1.0
net.floodlightcontroller.core.internal.OFSwitchManager.supportedOpenFlowVersions=1.0, 1.1, 1.2, 1.3, 1.4
//desde el mismo archivo de configuración es posible modificar el puerto en el cual el controlador escucha OF. El puerto por defecto es 6653
net.floodlightcontroller.core.internal.FloodlightProvider.openFlowPort=6653
```

Luego para iniciar el controlador, desde el directorio raíz de instalación ejecutar:

```
# java -jar target/floodlight.jar
```

Floodlight tiene un servicio web el cual permite entre otras funcionalidades, la visualización de la topología de red. Para acceder a dicho servicio, se debe ingresar en el navegador:

```
http://<ip_del_controlador>:8080/ui/pages/index.html
```

Luego si se accede a la pestaña correspondiente, se podrá visualizar la topología de red (Figura Anexo 2).

- Nota: se debe ejecutar el comando pingall desde mininet para que el controlador visualice la topología de red subyacente.

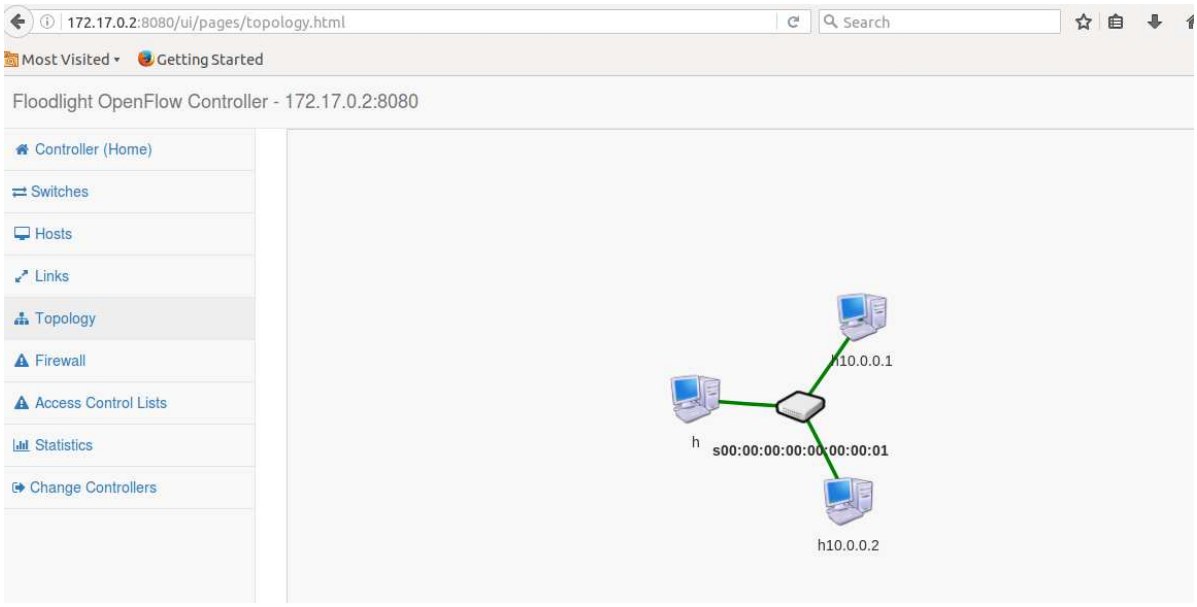


Figura Anexo 2 - Vista de topología de red ejemplo mediante Floodlight

7.ANEXO 7: SIMULACIÓN DE ATRIBUTO DE TRANSPARENCIA

i. *Configuración de red en Mininet:*

```
# sudo mn --controller=remote,ip=172.17.0.2 --custom=red_topo1.py --topo=redtopo --link=tc --mac
```

Se verifica la configuración desde la consola de mininet a través del comando net (Figura Anexo 3):

```
mininet> net
h1 h1-eth0:s3-eth3 h1-eth1:s4-eth3
h2 h2-eth0:s3-eth4 h2-eth1:s4-eth4
h3 h3-eth0:s5-eth3 h3-eth1:s6-eth3
h4 h4-eth0:s5-eth4 h4-eth1:s6-eth4
h5 h5-eth0:s1-eth7
h6 h6-eth0:s2-eth7
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s2-eth2 s1-eth3:s3-eth1 s1-eth4:s4-eth1 s1-eth5:
s5-eth1 s1-eth6:s6-eth1 s1-eth7:h5-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s1-eth2 s2-eth3:s3-eth2 s2-eth4:s4-eth2 s2-eth5:
s5-eth2 s2-eth6:s6-eth2 s2-eth7:h6-eth0
s3 lo:  s3-eth1:s1-eth3 s3-eth2:s2-eth3 s3-eth3:h1-eth0 s3-eth4:h2-eth0
s4 lo:  s4-eth1:s1-eth4 s4-eth2:s2-eth4 s4-eth3:h1-eth1 s4-eth4:h2-eth1
s5 lo:  s5-eth1:s1-eth5 s5-eth2:s2-eth5 s5-eth3:h3-eth0 s5-eth4:h4-eth0
s6 lo:  s6-eth1:s1-eth6 s6-eth2:s2-eth6 s6-eth3:h3-eth1 s6-eth4:h4-eth1
c0
```

Figura Anexo 3 - Transparencia - Configuración de red simulada

Dicha configuración se visualiza desde la interfaz de administración de OpenDayLight según se expone a continuación:

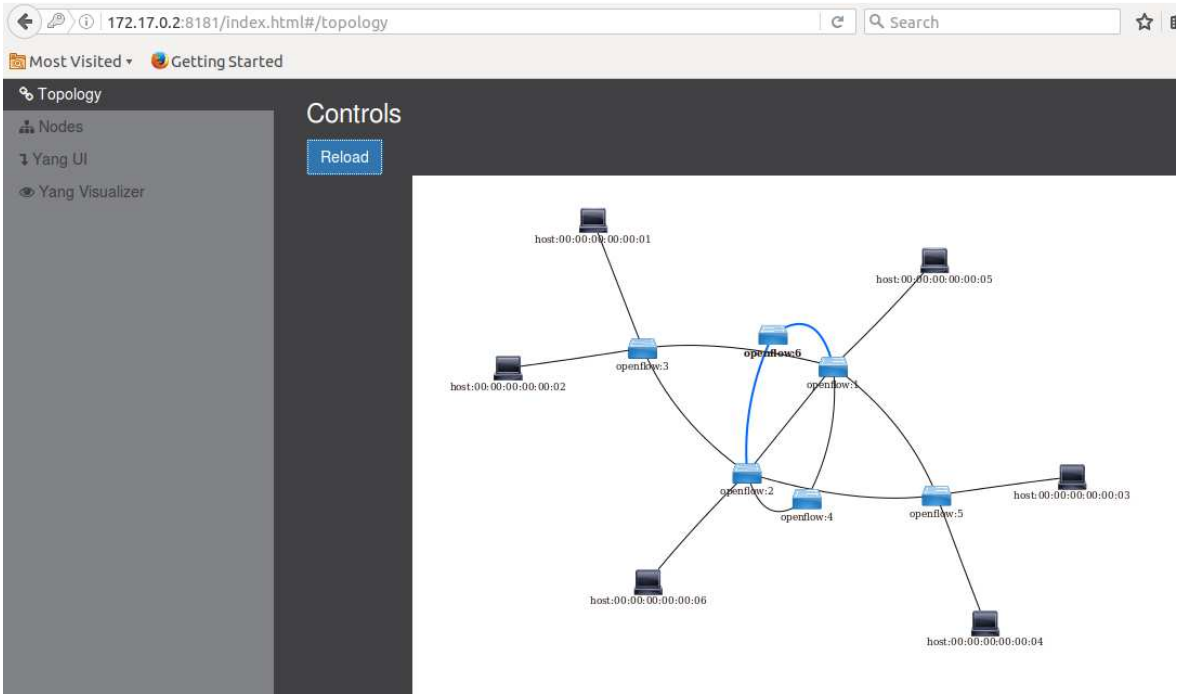
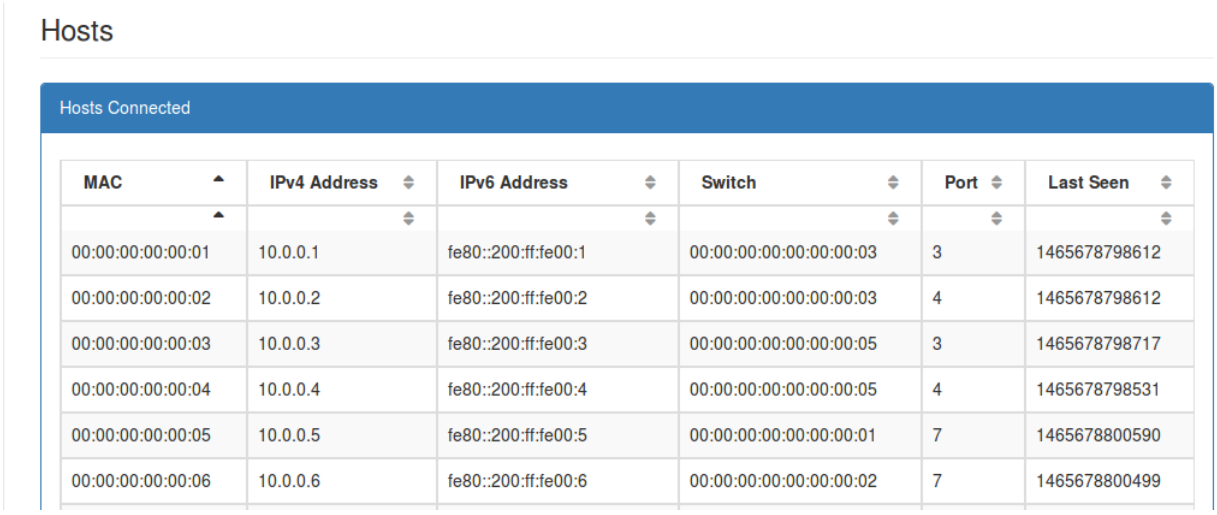


Figura Anexo 4 - Transparencia – Topología de red simulada desde OpenDayLight

Si asignamos el control a FloodLight, se visualizan desde su interfaz de administración los mismos dispositivos de red:



Hosts

Hosts Connected

MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen
00:00:00:00:00:01	10.0.0.1	fe80::200:ff:fe00:1	00:00:00:00:00:00:00:03	3	1465678798612
00:00:00:00:00:02	10.0.0.2	fe80::200:ff:fe00:2	00:00:00:00:00:00:00:03	4	1465678798612
00:00:00:00:00:03	10.0.0.3	fe80::200:ff:fe00:3	00:00:00:00:00:00:00:05	3	1465678798717
00:00:00:00:00:04	10.0.0.4	fe80::200:ff:fe00:4	00:00:00:00:00:00:00:05	4	1465678798531
00:00:00:00:00:05	10.0.0.5	fe80::200:ff:fe00:5	00:00:00:00:00:00:00:01	7	1465678800590
00:00:00:00:00:06	10.0.0.6	fe80::200:ff:fe00:6	00:00:00:00:00:00:00:02	7	1465678800499

Figura Anexo 5 – Transparencia – Host visualizados por Floodlight

Switches

Switches Connected

Switch ID	IPv4 Address	Connected Since
00:00:00:00:00:00:00:01	/172.17.0.1:59502	Sat Jun 11 2016 21:00:14 GMT-0300 (ART)
00:00:00:00:00:00:00:02	/172.17.0.1:59500	Sat Jun 11 2016 21:00:14 GMT-0300 (ART)
00:00:00:00:00:00:00:03	/172.17.0.1:59474	Sat Jun 11 2016 20:59:56 GMT-0300 (ART)
00:00:00:00:00:00:00:04	/172.17.0.1:59504	Sat Jun 11 2016 21:00:14 GMT-0300 (ART)
00:00:00:00:00:00:00:05	/172.17.0.1:59494	Sat Jun 11 2016 21:00:11 GMT-0300 (ART)
00:00:00:00:00:00:00:06	/172.17.0.1:59498	Sat Jun 11 2016 21:00:14 GMT-0300 (ART)

Showing 1 to 6 of 6 entries

Figura Anexo 6 - Transparencia – Switches adminstrados por Floodlight

ii. *Virtualización*

Se configura la red virtual “naranja” (Figura Anexo 7) en ambas soluciones de virtualización:

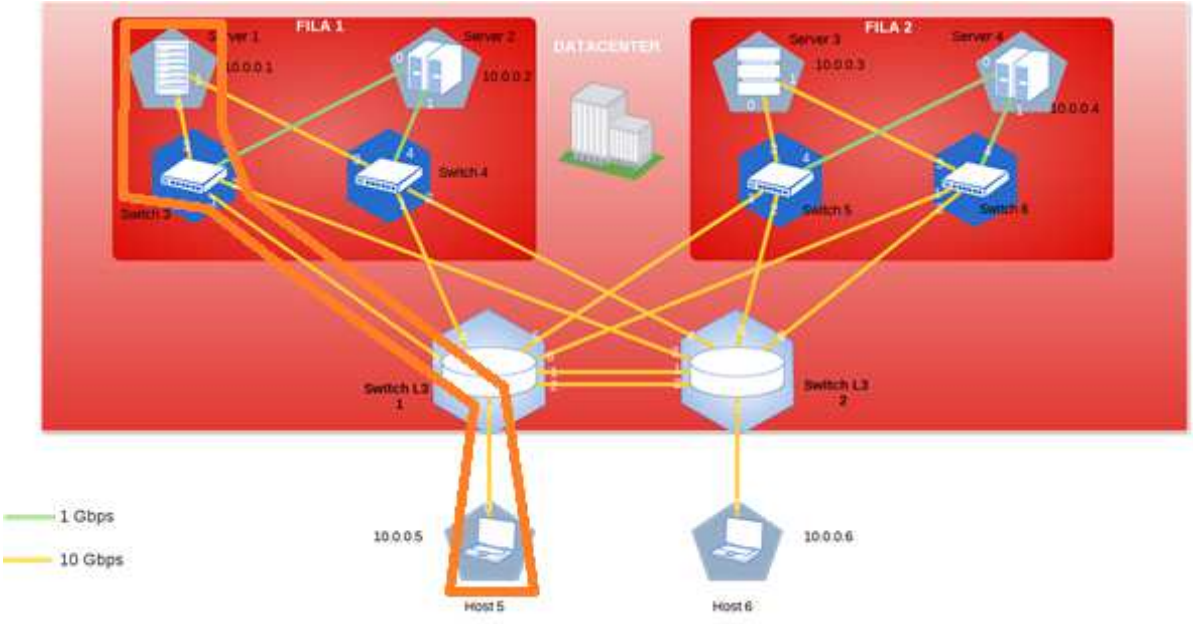


Figura Anexo 7 – Transparencia – Topología de red virtual Naranja

A. OpenVirtex

1) Creación de la red virtual

$ python ovxctl.py -n createNetwork tcp:172.17.0.3:10000 10.0.0.0 16

Virtual network has been created (network_id {u'mask': 16, u'networkAddress': 167772160, u'controllerUrls': [u'tcp:172.17.0.3:10000'], u'tenantId': 1}).

2) Creación de los switches virtuales

$ python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:03

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

$ python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:01

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02)

3) Creación de los puertos virtuales

$ python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 3

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

$ python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

$ python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 3

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 1)

$ python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 7

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 2)

4) Creación de links virtuales

$ python ovxctl.py -n connectLink 1 00:a4:23:05:00:00:00:01 2 00:a4:23:05:00:00:00:02 1 spf 1

Virtual link (link_id 1) has been created

5) Conexión de los host a los puertos virtuales

$ python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:01

Host (host_id 1) has been connected to virtual port

$ python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:02 2 00:00:00:00:00:05

Host (host_id 2) has been connected to virtual port

6) Inicio de la red virtual

$ python ovxctl.py -n startNetwork 1

Network (tenant_id 1) has been booted

Resultado

A través de la interfaz de Floodlight se visualiza la red virtual y en forma posterior el intercambio de pings entre los dos hosts correspondientes a la misma, verificando así su correcto funcionamiento:

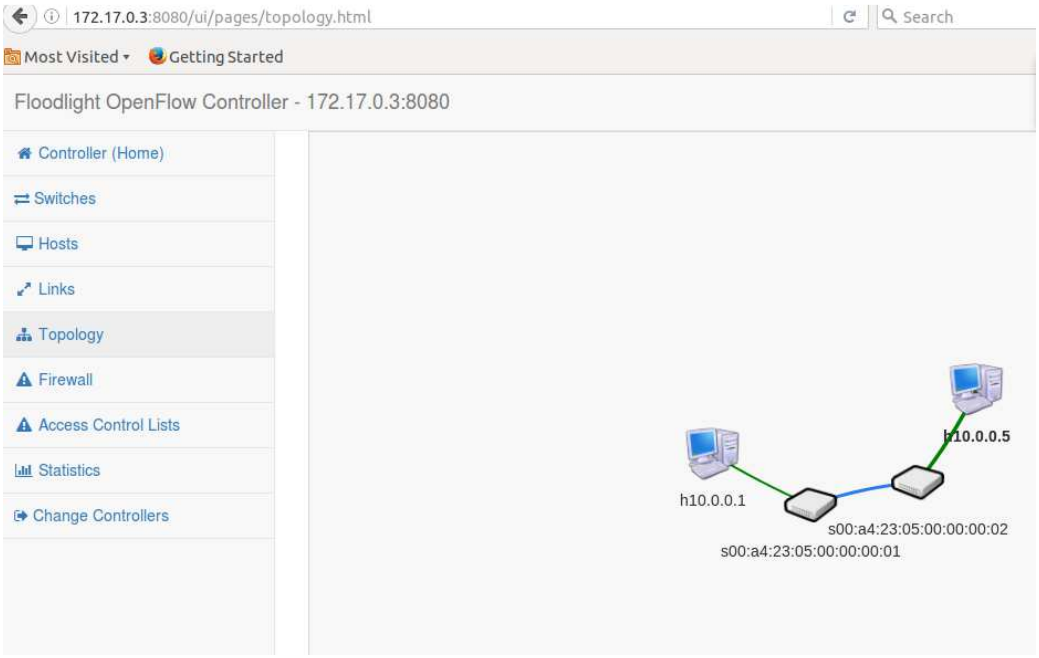


Figura Anexo 8 – Transparencia – Red virtual naranja virtualizada con OpenVirtex

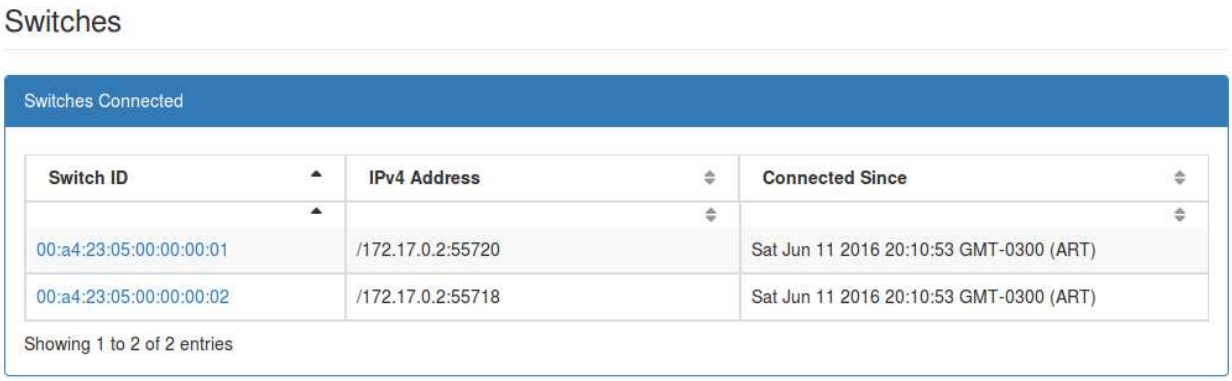


Figura Anexo 9 - Transparencia – Red virtual naranja virtualizada con OpenVirtex (Switches)

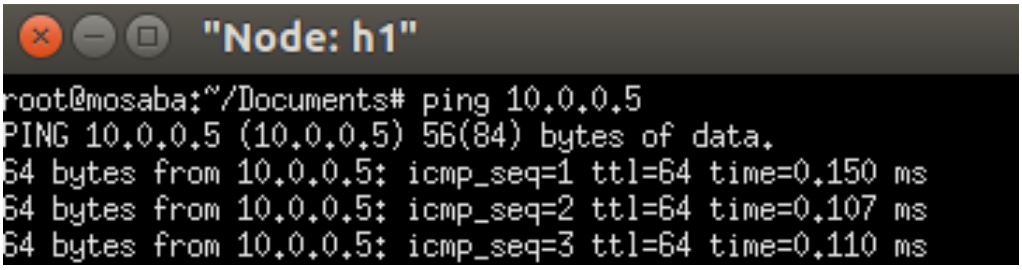


Figura Anexo 10 – Transparencia – ping desde host 1 a host 5

B. Flowvisor

1) Generación de red y asignación al controlador:

fvctl -f /dev/null add-slice naranja tcp:172.17.0.3:10000 admin@naranja
Slice password:
Slice naranja was successfully created

2) Configuración de espacios de flujo:

fvctl -f /dev/null add-flowspace dpid3-port3 3 1 in_port=3 naranja=7
fvctl -f /dev/null add-flowspace dpid3-port1 3 1 in_port=1 naranja=7
FlowSpace dpid3-port1 was added with request id 1.
fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3 naranja=7
FlowSpace dpid1-port3 was added with request id 3.
fvctl -f /dev/null add-flowspace dpid1-port7 1 1 in_port=7 naranja=7
FlowSpace dpid1-port7 was added with request id 4.

3) Verificación de configuración:

#fvctl -f /dev/null list-slices

Slices creados:

Configured slices:
fvadmin --> enabled
naranja --> enabled
fvctl -f /dev/null list-flowspace
Configured Flow entries:
{"force-enqueue": -1, "name": "dpid3-port3", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:03", "id": 1, "match": {"wildcards": 4194302, "in_port": 3}}

{"force-enqueue": -1, "name": "dpid3-port1", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:03", "id": 2, "match": {"wildcards": 4194302, "in_port": 1}}

{"force-enqueue": -1, "name": "dpid1-port3", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:01", "id": 4, "match": {"wildcards": 4194302, "in_port": 3}}

{"force-enqueue": -1, "name": "dpid1-port7", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:01", "id": 5, "match": {"wildcards": 4194302, "in_port": 7}}

Resultado

A través de la interfaz de Floodlight se visualiza la red virtual y en forma posterior el intercambio de pings entre los dos hosts correspondientes a la misma, verificando así su correcto funcionamiento:

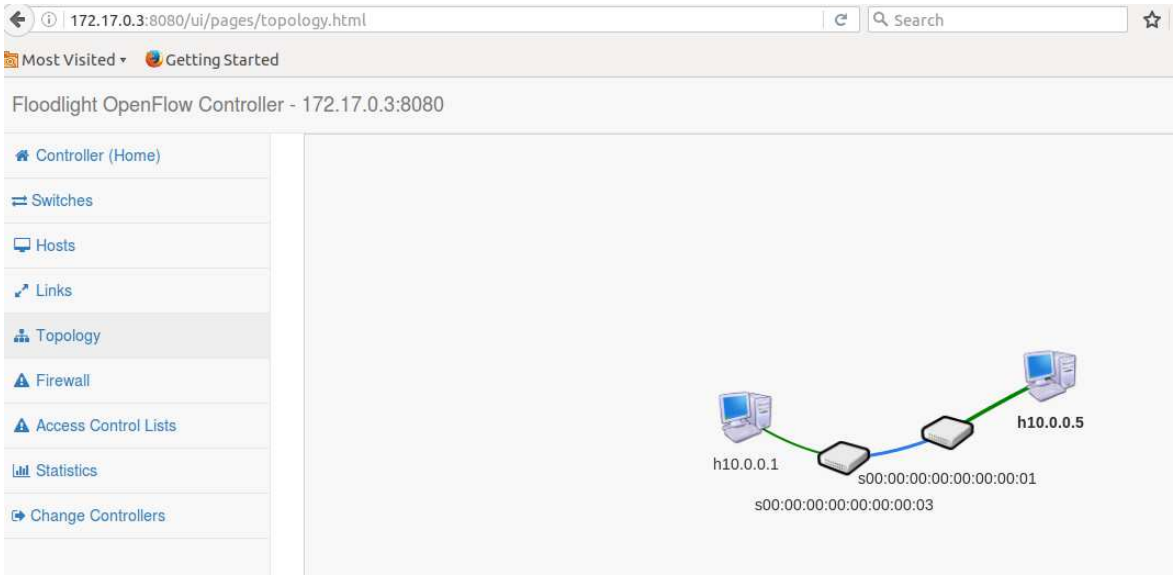


Figura Anexo 11 – Transparencia - Red virtual naranja virtualizada con FlowVisor

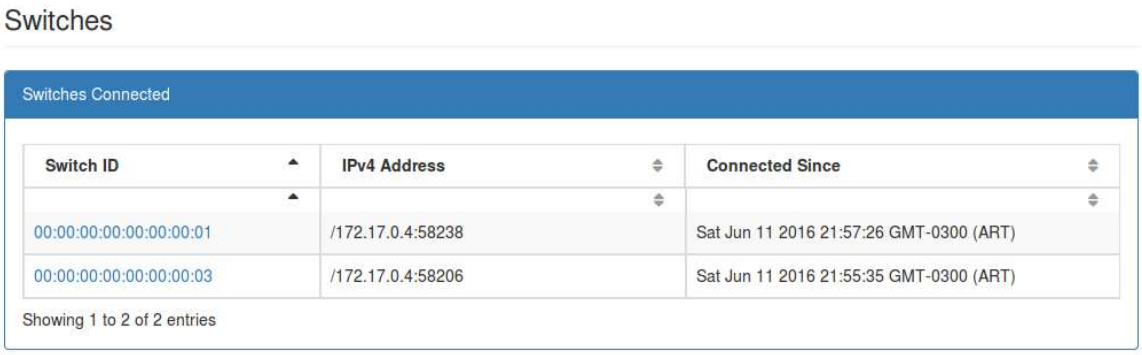


Figura Anexo 12 - Transparencia – Red virtual naranja virtualizada con FlowVisor (Switches)

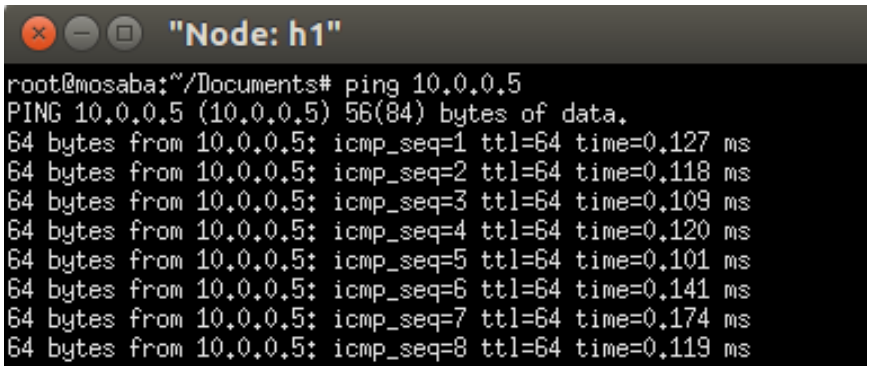


Figura Anexo 13 - Transparencia – ping desde host 1 a host 5

8.ANEXO 8: SIMULACIÓN DE ATRIBUTO DE AISLACIÓN

Plano de Control

i. *Configuración de red en Mininet:*

La configuración de red es la misma que la utilizada en ANEXO 7: Simulación de atributo de transparencia.

ii. *Virtualización*

Se configura la red virtual “naranja” al igual que en el ANEXO 7: Simulación de atributo de transparencia y la red “gris”. Ambas redes se exponen en la Figura Anexo 14.

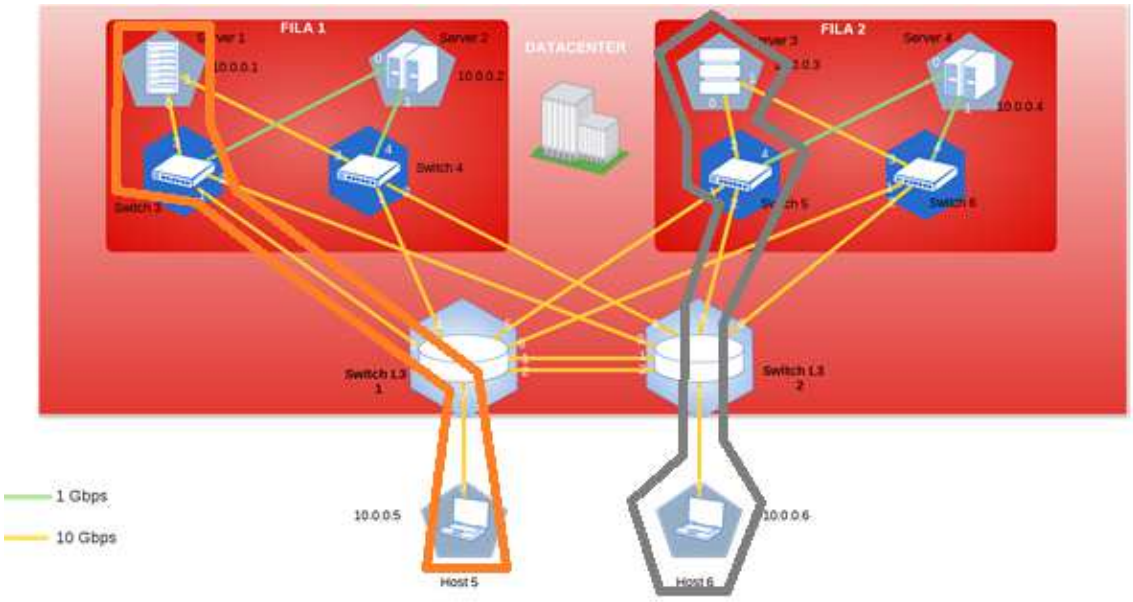


Figura Anexo 14 – Aislación – Topología red naranja y gris

La configuración de virtualización de la red naranja, se realizó en la simulación del atributo de transparencia. A continuación, se exponen las configuraciones para la red gris en ambas soluciones de virtualización.

A. OpenVirtex

1) Creación de la red virtual Gris

$ python ovxctl.py -n createNetwork tcp:172.17.0.4:20000 10.0.0.0 16

Virtual network has been created (network_id {u'mask': 16, u'networkAddress': 167772160, u'controllerUrls': [u'tcp:172.17.0.4:20000'], u'tenantId': 2}).

2) Creación de los switches virtuales

$ python ovxctl.py -n createSwitch 2 00:00:00:00:00:00:00:05

Virtual switch has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01)

$ python ovxctl.py -n createSwitch 2 00:00:00:00:00:00:00:02

Virtual switch has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02)

3) Creación de los puertos virtuales

$ python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:05 2

Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

$ python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:05 3

Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

$ python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:02 5

Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02, port_id 1)

$ python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:02 7

Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02, port_id 2)

4) Creación de links virtuales

$ python ovxctl.py -n connectLink 2 00:a4:23:05:00:00:00:01 1 00:a4:23:05:00:00:00:02 1 sfp 1

Virtual link (link_id 1) has been created

5) Conexión de los hosts a los puertos virtuales

$ python ovxctl.py -n connectHost 2 00:a4:23:05:00:00:00:01 2 00:00:00:00:00:03

Host (host_id 1) has been connected to virtual port

$ python ovxctl.py -n connectHost 2 00:a4:23:05:00:00:00:02 2 00:00:00:00:00:06

Host (host_id 2) has been connected to virtual port

6) Inicio de la red virtual

$ python ovxctl.py -n startNetwork 2

Network (tenant_id 2) has been booted

Resultado

Se procede a visualizar la topología desde la gestión del controlador 1 (red naranja):

Figura Anexo 15 – Aislación – Topología de red Naranja (Controlador 1)

El mismo procedimiento se realiza desde la interfaz de administración del controlador 2 (red gris):

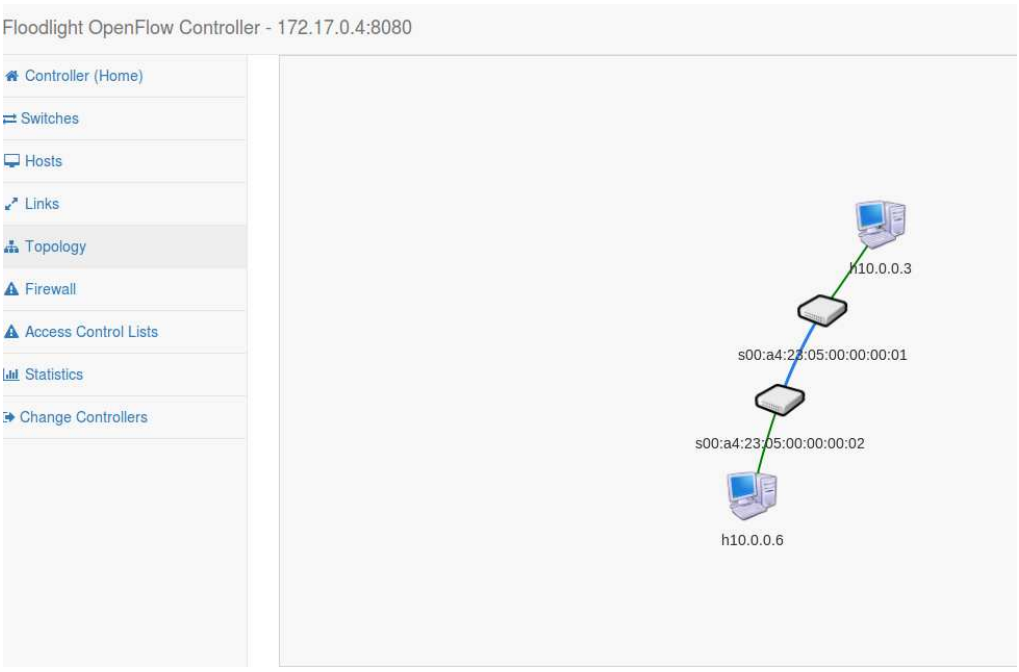


Figura Anexo 16 - Aislación – Topología de red Gris (Controlador 2)

B. Flowvisor

Configuración de la red virtual gris:

Para la configuración de la siguiente red virtual, se utiliza el API de administración y configuración de FV fvctl.

1) Configuración de la red virtual gris y asignación al controlador correspondiente:

```
# fvctl -f /dev/null add-slice gris tcp:172.17.0.4:20000 admin@gris
Slice password:
Slice gris was successfully created
```

2) Configuración del slice o segmento de red correspondiente a gris (en este caso la segmentación se da por puerto físico de entrada):

```
# fvctl -f /dev/null add-flowspace dpid5-port3 5 1 in_port=3 gris=7
FlowSpace dpid5-port3 was added with request id 4.
```

```
# fvctl -f /dev/null add-flowspace dpid5-port2 5 1 in_port=2 gris=7
FlowSpace dpid5-port2 was added with request id 5.
# fvctl -f /dev/null add-flowspace dpid2-port5 2 1 in_port=5 gris=7
FlowSpace dpid2-port5 was added with request id 6.
# fvctl -f /dev/null add-flowspace dpid2-port7 2 1 in_port=7 gris=7
FlowSpace dpid2-port7 was added with request id 7.
```

Resultado

Se procede a visualizar la topología desde la gestión del controlador 1 (red naranja):

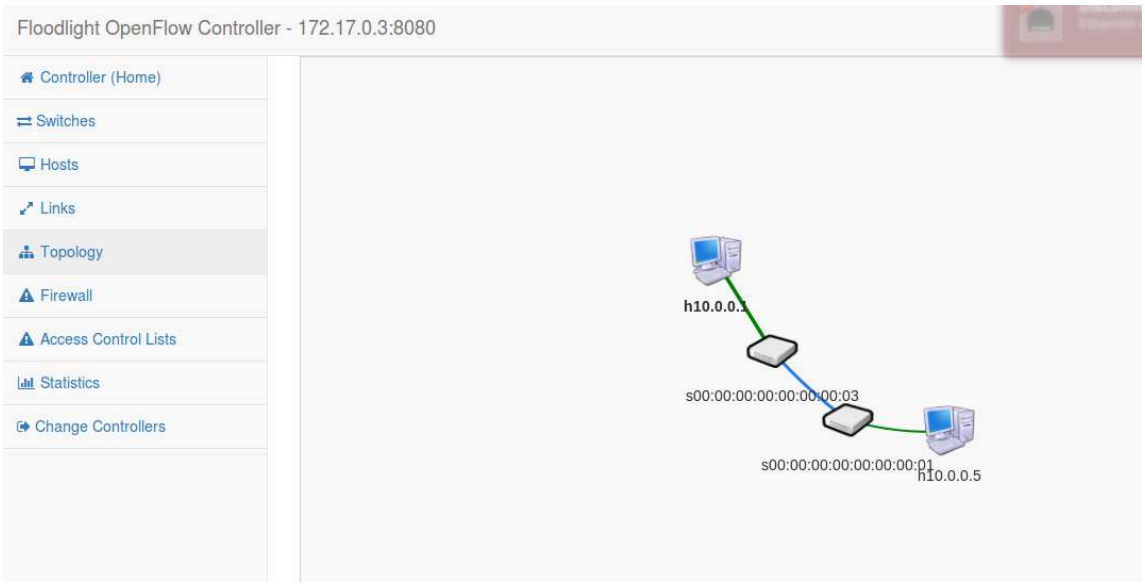


Figura Anexo 17 - Aislación – Topología de red Naranja (Controlador 1)

Se procede a visualizar la topología desde la gestión del controlador 2 (red gris):

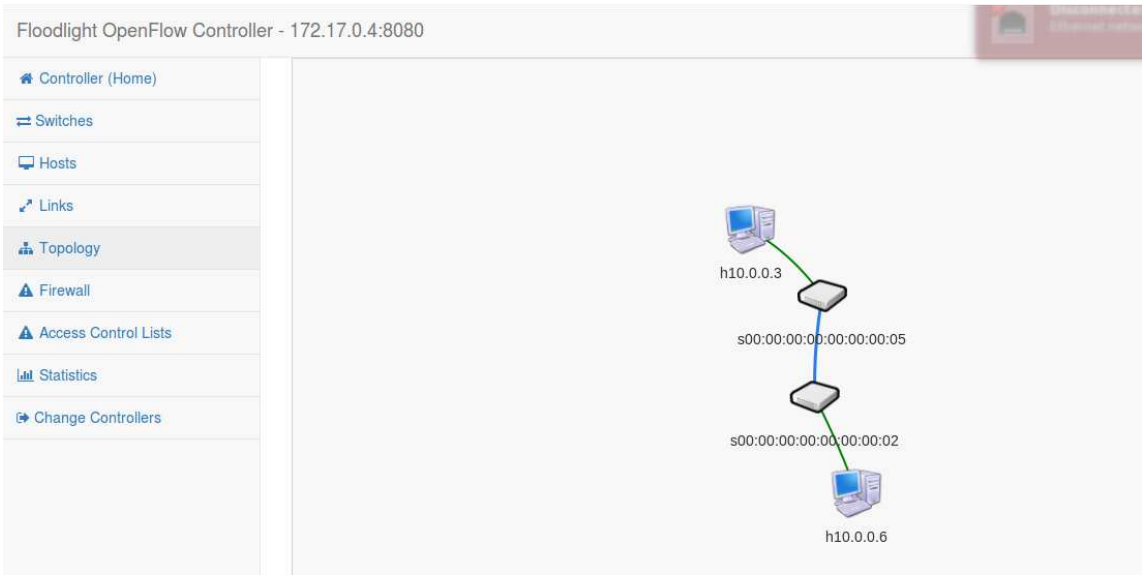


Figura Anexo 18 - Aislación – Topología de red Gris (Controlador 2)

Adicionalmente podremos visualizar la comunicación entre los hosts correspondientes a cada red virtual. Host 1 con el host 5 y host 3 con el host 6 (Figura Anexo 14 – Aislación – Topología red naranja y gris):

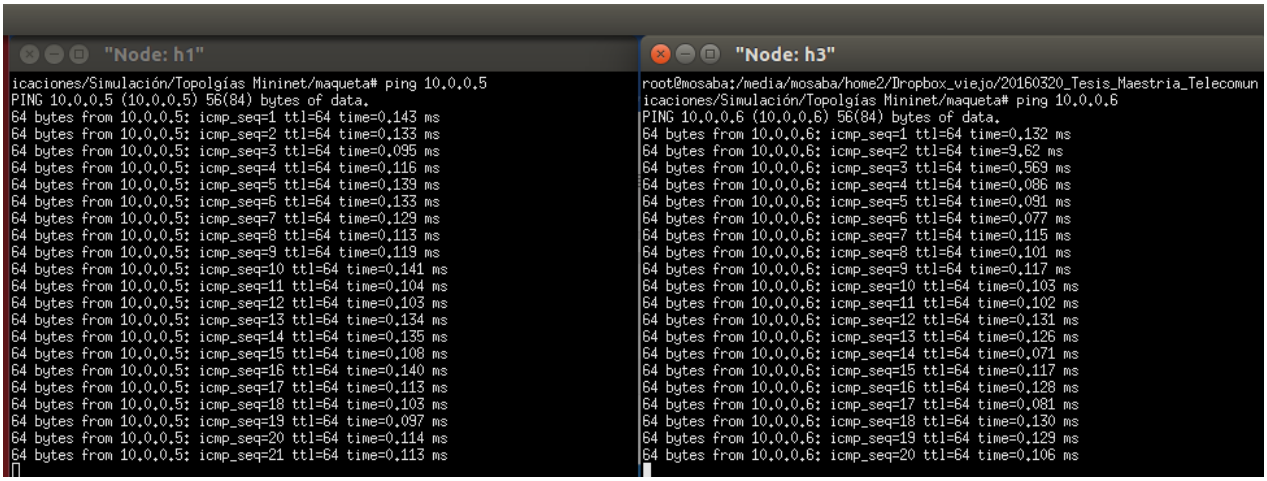


Figura Anexo 19 – Aislación – Resultado ping entre h1/h5 y h3/h6

De manera contraria, no existe comunicación entre los hosts correspondientes a distintos segmentos de red, lo cual permite graficar la aislación entre las dos redes virtuales.

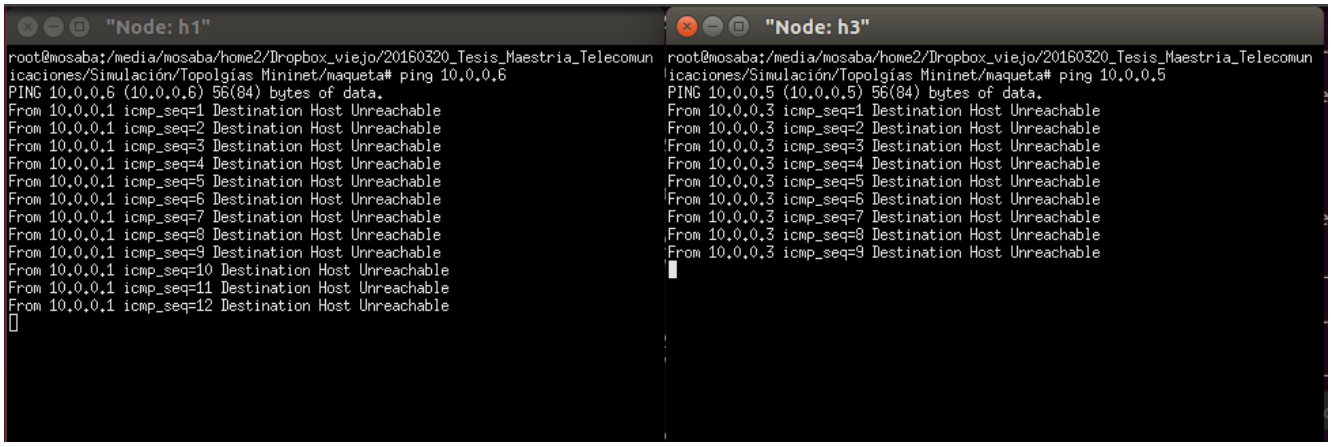


Figura Anexo 20 – Aislación – Resultado ping entre h1 y h3

Aislación de los esquemas de direccionamiento

Se procede a asignar a los hosts 3 y 5 de la red virtual gris las mismas direcciones IP que los hosts 1 y 5 de la red naranja:

Para ello en mininet se ejecutan los siguientes comandos:

mininet> h3 ifconfig h3-eth0 10.0.0.1 netmask 255.0.0.0
mininet> h6 ifconfig h6-eth0 10.0.0.5 netmask 255.0.0.0

El host 3 pasará a tener la misma dirección IP que el host 1, mientras que el host 6 la misma que la del host 5 (Figura Anexo 21).

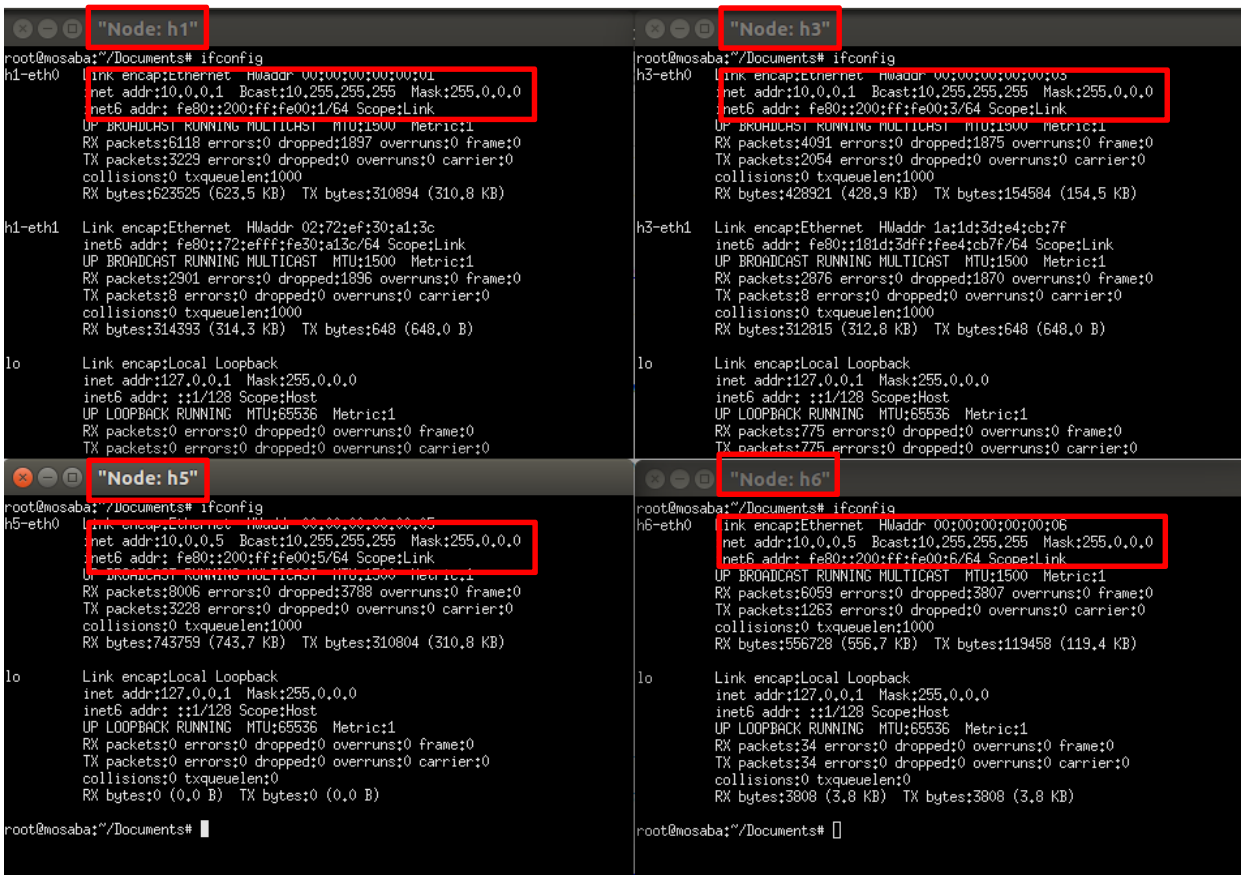


Figura Anexo 21- Aislación – Esquema de direccionamiento superpuesto en red virtual gris y naranja.

A. OpenVirtex

Si se realiza ping desde el host 1 al 5 y del 3 al 6 en forma simultánea, veremos que es posible superponer el mismo esquema de direccionamiento en dos redes virtuales sobre la misma red física (Figura Anexo 22).

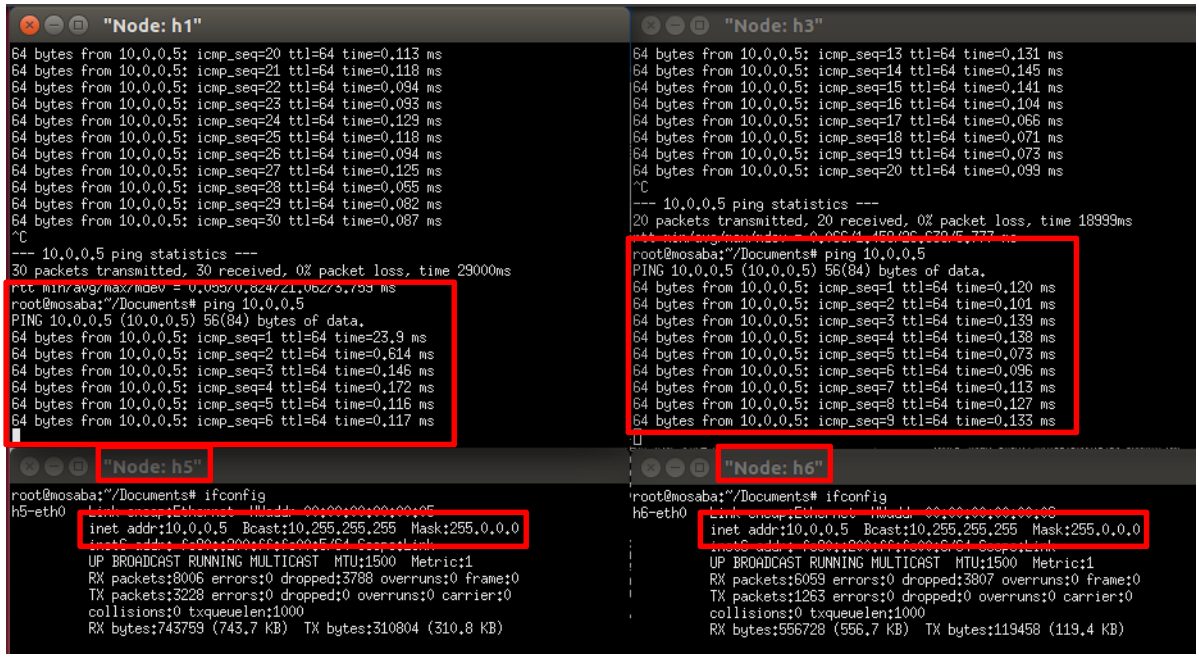


Figura Anexo 22 – Aislación – ping host 1 al 5 y del 3 al 6 en forma simultánea

Se ejecuta el siguiente comando en mininet, para ver las tablas de flujo de los switches OVS simulados:

mininet>dpctl dump-flows

Switches de red GRIS:

*** s5 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x200000001, duration=264.364s, table=0, n_packets=262, n_bytes=25676, idle_timeout=5, idle_age=1, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:06,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=mod_nw_dst:2.0.0.5,mod_nw_src:2.0.0.4,mod_dl_src:a4:23:05:02:00:00,mod_dl_dst:a4:23:05:10:00:02,output:2

cookie=0x200000003, duration=264.352s, table=0, n_packets=262, n_bytes=25676, idle_timeout=5, idle_age=1, priority=1,ip,in_port=2,dl_src=a4:23:05:02:00:00,dl_dst=a4:23:05:10:00:04,nw_src=2.0.0.5,nw_dst=2.0.0.4 actions=mod_nw_src:10.0.0.5,mod_nw_dst:10.0.0.1,mod_dl_src:00:00:00:00:00:06,mod_dl_dst:00:00:00:00:00:03,output:3

*** s2 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x200000004, duration=4.683s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, idle_age=4, priority=1,arp,in_port=7,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03 actions=mod_dl_src:a4:23:05:02:00:00,mod_dl_dst:a4:23:05:10:00:04,output:5

cookie=0x200000002, duration=4.681s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, idle_age=4, priority=1,arp,in_port=5,dl_src=a4:23:05:02:00:00,dl_dst=a4:23:05:10:00:02 actions=mod_dl_src:00:00:00:00:00:03,mod_dl_dst:00:00:00:00:00:06,output:7

cookie=0x200000001, duration=147.935s, table=0, n_packets=146, n_bytes=14308, idle_timeout=5, idle_age=0, priority=1,ip,in_port=5,dl_src=a4:23:05:02:00:00,dl_dst=a4:23:05:10:00:02,nw_src=2.0.0.4,nw_dst=2.0.0.5 actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.5,mod_dl_src:00:00:00:00:00:03,mod_dl_dst:00:00:00:00:00:06,output:7

cookie=0x200000003, duration=147.922s, table=0, n_packets=146, n_bytes=14308, idle_timeout=5, idle_age=0, priority=1,ip,in_port=7,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=mod_nw_dst:2.0.0.4,mod_nw_src:2.0.0.5,mod_dl_src:a4:23:05:02:00:00,mod_dl_dst:a4:23:05:10:00:04,output:5

Switches de red NARANJA:

*** s3 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x100000001, duration=617.22s, table=0, n_packets=616, n_bytes=60368, idle_timeout=5, idle_age=1, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,mod_dl_src:a4:23:05:01:00:00,mod_dl_dst:a4:23:05:10:00:02,output:1

cookie=0x100000004, duration=617.217s, table=0, n_packets=616, n_bytes=60368, idle_timeout=5, idle_age=1, priority=1,ip,in_port=1,dl_src=a4:23:05:01:00:00,dl_dst=a4:23:05:10:00:04,nw_src=1.0.0.2,nw_dst=1.0.0.1 actions=mod_nw_src:10.0.0.5,mod_nw_dst:10.0.0.1,mod_dl_src:00:00:00:00:00:05,mod_dl_dst:00:00:00:00:00:01,output:3

*** s1 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x100000004, duration=617.195s, table=0, n_packets=616, n_bytes=60368, idle_timeout=5, idle_age=1, priority=1,ip,in_port=7,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=mod_nw_dst:1.0.0.1,mod_nw_src:1.0.0.2,mod_dl_src:a4:23:05:01:00:00,mod_dl_dst:a4:23:05:10:00:04,output:3

cookie=0x100000001, duration=617.214s, table=0, n_packets=616, n_bytes=60368, idle_timeout=5, idle_age=1, priority=1,ip,in_port=3,dl_src=a4:23:05:01:00:00,dl_dst=a4:23:05:10:00:02,nw_src=1.0.0.1,nw_dst=1.0.0.2 actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.5,mod_dl_src:00:00:00:00:00:01,mod_dl_dst:00:00:00:00:00:05,output:7

Analizando las tablas anteriores, se ve cómo funciona OVS para lograr la aislación en el direccionamiento. OVX escribe las tablas de flujo para modificar en los switches de borde, las direcciones de origen y destino (Figura Anexo 23).

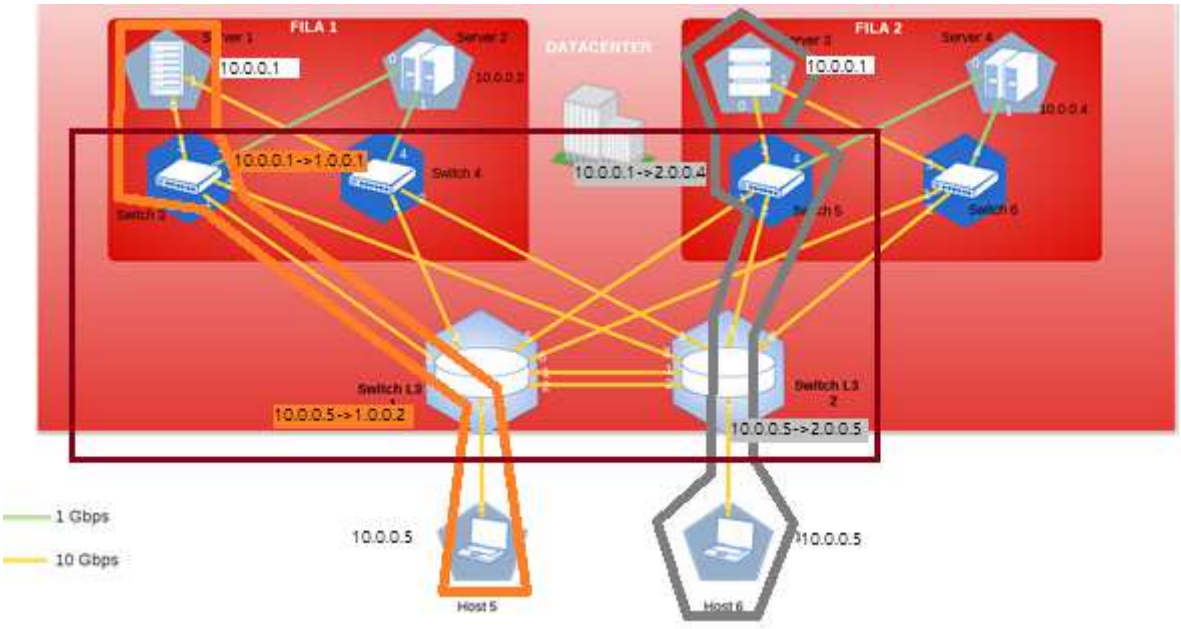


Figura Anexo 23 – Aislación – Mecanismo de Superposición de direccionamiento IP OpenVirtex

Resultado

Los switches de borde de la red modifican las direcciones de IP origen y destino dentro de la red simulada. Luego de que los paquetes son direccionados según corresponda dentro de la red, otro switch de borde tiene la responsabilidad de volver a asignar las direcciones originales, para que éstos sean entregados a los hosts según corresponda. Ello permite la superposición de esquemas de direccionamiento en dos redes virtuales corriendo sobre la misma red física.

B. FlowVisor

Utilizando una configuración análoga a la implementada en el caso de OVX, tanto el host 1 como el 3 tienen la dirección ip 10.0.0.1, mientras que el host 5 y 6 comparten la 10.0.0.5 (Figura Anexo 24).

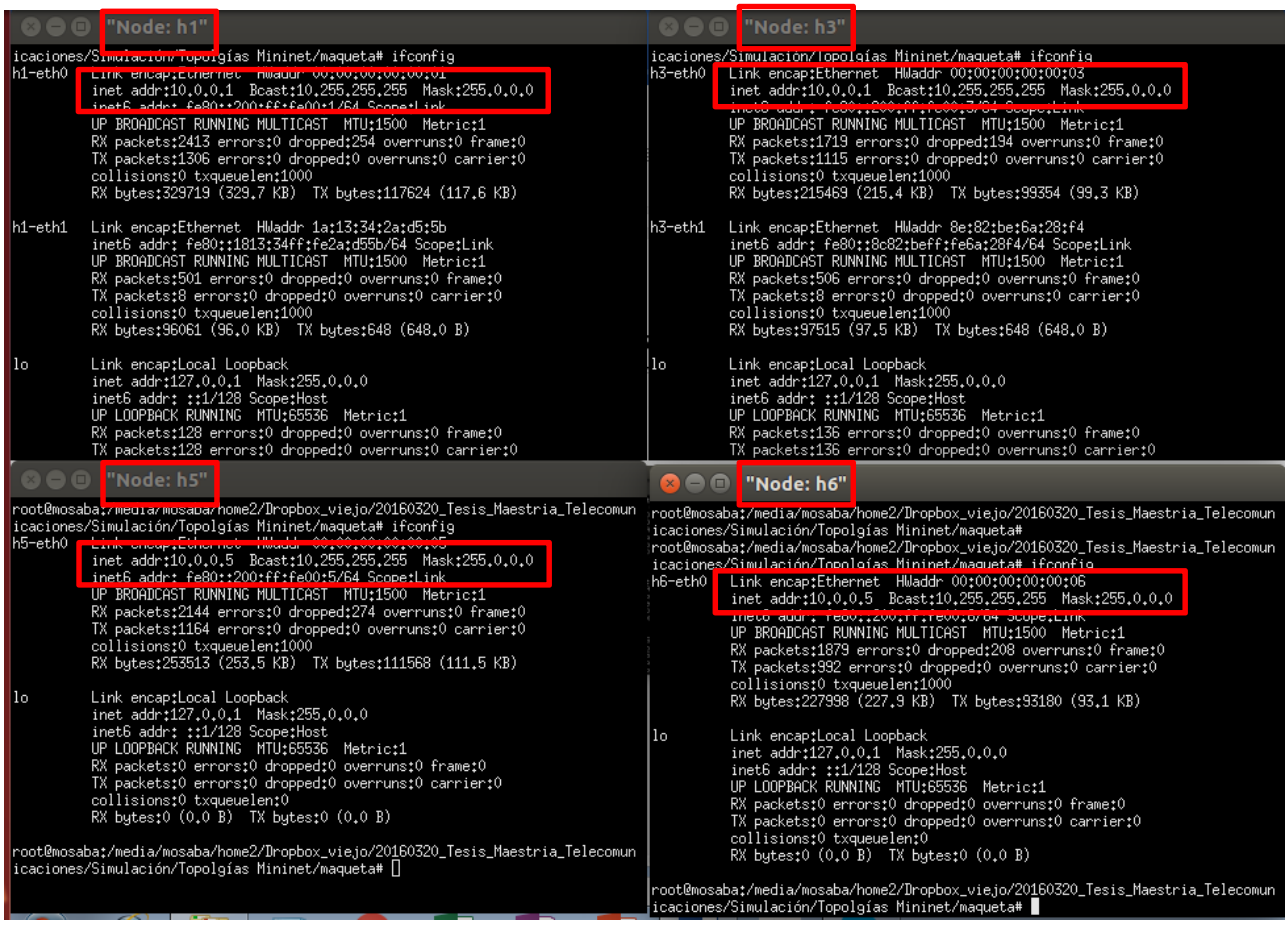


Figura Anexo 24 – Aislación – Configuración de direcciones IP en los hosts

Resultado

Si se realiza el proceso análogo efectuado con OVX, ejecutando ping desde el host 1 al 5 y del 3 al 6 en forma simultánea, veremos que NO es posible superponer en todos los casos el mismo esquema de direccionamiento en dos redes virtuales sobre la misma red física.

Esto se debe a que se podría dar el caso en el cual ambos slices compartan el mismo espacio de flujos y FV sólo mantiene la aislación mientras que los espacios de flujo no se superpongan entre sí.

Esta situación se describe en [33] - “FV puede aislar dos slices en el caso que se asegure que sus espacios de flujo no se superpongan en ningún lugar en la topología de red”.

Si se analizan las tablas de flujo en los switches físicos, se podrá apreciar que FV no posee ningún mecanismo de reescritura para posibilitar la superposición de direccionamiento, lo cual se corresponde con lo expresado en [33]. En este punto FV se diferencia de OVX en el sentido de que el segundo reescribe los flujos en el interior de la red para garantizar la superposición de direccionamiento.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x146, duration=2823.073s, table=0, n_packets=2822, n_bytes=276556, idle_timeout=5, idle_age=2, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:7
 cookie=0x147, duration=2823.067s, table=0, n_packets=2822, n_bytes=276556, idle_timeout=5, idle_age=1, priority=1,ip,in_port=7,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=output:3
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x151, duration=2811.687s, table=0, n_packets=2811, n_bytes=275478, idle_timeout=5, idle_age=0,
```

priority=1,ip,in_port=7,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=output:5

cookie=0x150, duration=2811.692s, table=0, n_packets=2812, n_bytes=275576, idle_timeout=5, idle_age=0, priority=1,ip,in_port=5,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:06,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:7

*** s3 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x146, duration=2823.085s, table=0, n_packets=2822, n_bytes=276556, idle_timeout=5, idle_age=1, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:1

cookie=0x147, duration=2823.081s, table=0, n_packets=2823, n_bytes=276654, idle_timeout=5, idle_age=1, priority=1,ip,in_port=1,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=output:3

*** s4 ----------

NXST_FLOW reply (xid=0x4):

*** s5 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x150, duration=2811.708s, table=0, n_packets=2811, n_bytes=275478, idle_timeout=5, idle_age=0, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:06,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:2

cookie=0x151, duration=2811.703s, table=0, n_packets=2812, n_bytes=275576, idle_timeout=5, idle_age=0, priority=1,ip,in_port=2,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=output:3

*** s6 ----------

NXST_FLOW reply (xid=0x4):

De manera gráfica, se estaría dando la siguiente situación en la cual en el interior de la red los flujos conservan las direcciones de IP destino y origen de la red física.

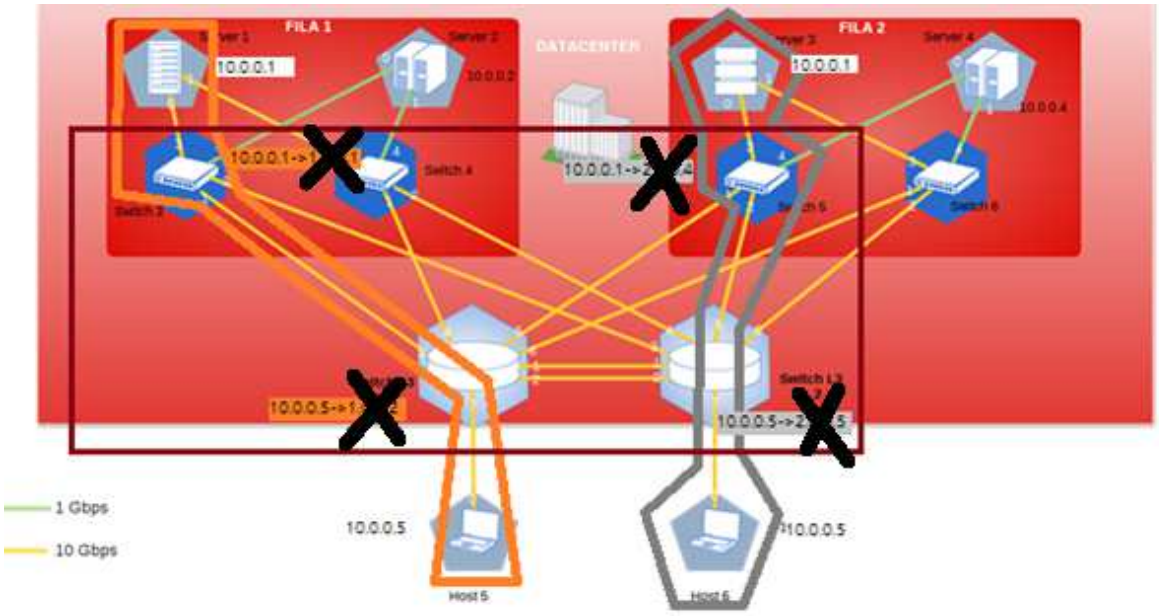


Figura Anexo 25 – Aislación - Superposición de direccionamiento en dos redes virtuales FV

Tal como se expresó, dicha situación no garantiza que la definición de dos flujos no se superponga en un switch perjudicando así la aislación.

Para comprender mejor este concepto, se presenta la red siguiente (Figura Anexo 26) en donde tanto el host 1 como el 3 comparten la misma dirección IP (10.0.0.1).

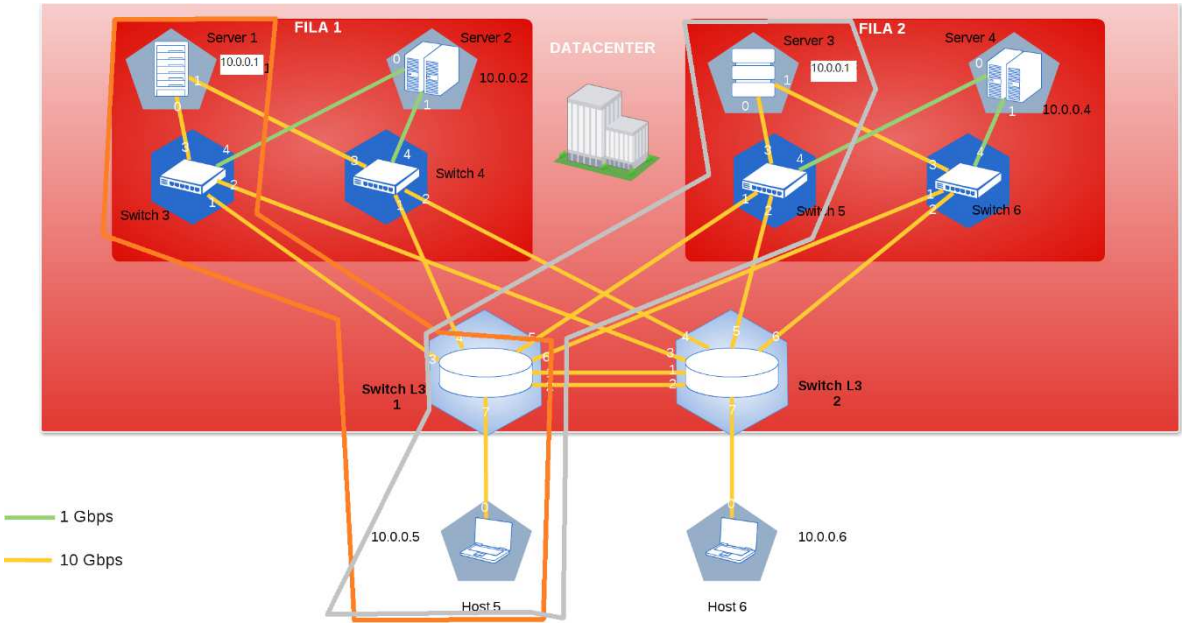


Figura Anexo 26 – Aislación – Red gris modificada

Tal como se puede apreciar, ambas redes comparten el switch L3 1. Tal como se vió en 1.SDN apartado c., la versión 1.0 de OF soporta la definición de flujos utilizando doce campos (Figura 8).

Para simplificar el ejemplo, supongamos que la definición de flujos está limitada solo a tres campos: puerto de ingreso, ip de origen e ip de destino. En este caso, la tabla de flujos correspondiente al switch L3 1 debiera ser similar a la siguiente:

*** s1 ----------

NXST_FLOW reply (xid=0x4):

……in_port=5,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:7…….

……in_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.5 actions=output:7…….

……in_port=7,nw_src=10.0.0.5,nw_dst=10.0.0.1 actions=output: 3 o 5

Se resalta en rojo la problemática que se presenta, la cual expone, una de las limitaciones de FV. En este caso el switch no sabría porque puerto debería enviar el tráfico cuyo origen se presenta en el h5 (IP 10.0.0.5) y tiene como destino el h1/3 (IP 10.0.0.1).

Dicho ejemplo muestra la superposición en la definición de los flujos asociados a cada slice. Se expone en este caso, acotando el espacio de flujos solo a tres campos, de manera tal de clarificar este concepto. Sin embargo, el mismo caso se podría dar utilizando la totalidad de los campos definidos en OF 1.0.

Es por ello que, si bien FV no impide la utilización del mismo esquema de direccionamiento en dos redes virtuales, no se garantiza la correcta aislación puesto que pudiera darse el caso expuesto en el ejemplo precedente.

9. ANEXO 9: SIMULACIÓN DE ATRIBUTO DE AISLACIÓN DE TRÁFICO (PLANO DE DATOS)

i. *Configuración de red en Mininet:*

La configuración de red es la misma que la utilizada en ANEXO 7: Simulación de atributo de transparencia.

ii. *Virtualización*

Se configura la red virtual “naranja” al igual que en el ANEXO 7: Simulación de atributo de transparencia y otra red virtual análoga desde el punto de vista de topología (interconecta los host 1 y host 5) pero solamente con control sobre el tráfico TCP puerto 555. A este último slice se lo denomina puerto_prueba.

La red naranja se asigna al controlador cuya IP y puerto es 172.17.0.3:10000.

Luego, se configura la red puerto_prueba asignándole el control a un segundo controlador cuya IP y puerto son: 172.17.0.4:20000.

A. FlowVisor

La configuración de la red naranja es análoga a la expuesta en ANEXO 7: Simulación de atributo de transparencia.

A continuación, se expone la configuración de la red puerto_prueba:

1) Creación de red virtual puerto_prueba y asignación al controlador:

```
# fvctl -f /dev/null add-slice puerto_prueba tcp:172.17.0.4:20000 admin@puerto_pruebaslice
Slice password:
Slice puerto_prueba was successfully created
```

2) Creación de los espacios de flujo que manejará la red puerto prueba:

```
# fvctl -f /dev/null add-flowspace dpid3-port3-puerto-src 3 100 in_port=3,dl_type=0x0800,nw_proto=6,tp_src=555 puerto_prueba=7
# fvctl -f /dev/null add-flowspace dpid3-port3-puerto-dst 3 100 in_port=3,dl_type=0x0800,nw_proto=6,tp_dst=555 puerto_prueba=7
FlowSpace dpid3-port3-puerto-dst was added with request id 1.
# fvctl -f /dev/null add-flowspace dpid1-port7-puerto-src 1 100 in_port=7,dl_type=0x0800,nw_proto=6,tp_src=555 puerto_prueba=7
FlowSpace dpid1-port7-puerto-src was added with request id 2.
```

fvctl -f /dev/null add-flowspace dpid1-port7-puerto-dst 1 100 in_port=7,dl_type=0x0800,nw_proto=6,tp_dst=555 puerto_prueba=7

FlowSpace dpid1-port7-puerto-dst was added with request id 3.

fvctl -f /dev/null add-flowspace dpid3-port1-puerto 3 100 in_port=1 puerto_prueba=7

FlowSpace dpid3-port1-puerto was added with request id 4.

fvctl -f /dev/null add-flowspace dpid1-port3-puerto 1 100 in_port=3 puerto_prueba=7

FlowSpace dpid1-port3-puerto was added with request id 5.

De acuerdo a la configuración, se puede apreciar que, por la definición de los espacios de flujo, la red puerto_prueba sólo manejará el tráfico TCP puerto 555 entre los hosts 1 y 5 tal como se había propuesto. Se debe recordar, que el controlador asociado a la red naranja, controla todo el tráfico incluyendo el asociado al puerto 555.

Para probar que efectivamente el controlador puerto_prueba sólo controla el puerto 555, se utiliza iperf. Esta herramienta permite probar flujos de datos en diversos puertos.

Resultado

Mientras están ambos controladores funcionando, es posible generar flujos de tráfico en todos los puertos entre h1 y h5.

A continuación, se muestran algunos ejemplos:

1) Se realiza un ping entre h1 y h5 (Figura Anexo 27)

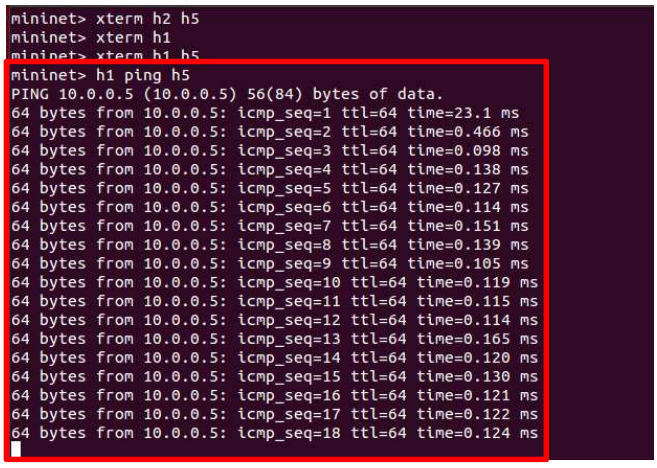


Figura Anexo 27- Aislación de tráfico – ping entre host 1 y 5

2) Se verifica la conectividad en el puerto 8080 (http) entre ambos (Figura Anexo 28).

Se configura en el host 1 iperf para que escuche en el puerto 8080:

```
# iperf -s –p 8080
```

Se configura el host 5 para que envíe tráfico al host 1 en dicho puerto:

```
# iperf -c 10.0.0.1 -p 8080 -t 10 -i 1
```

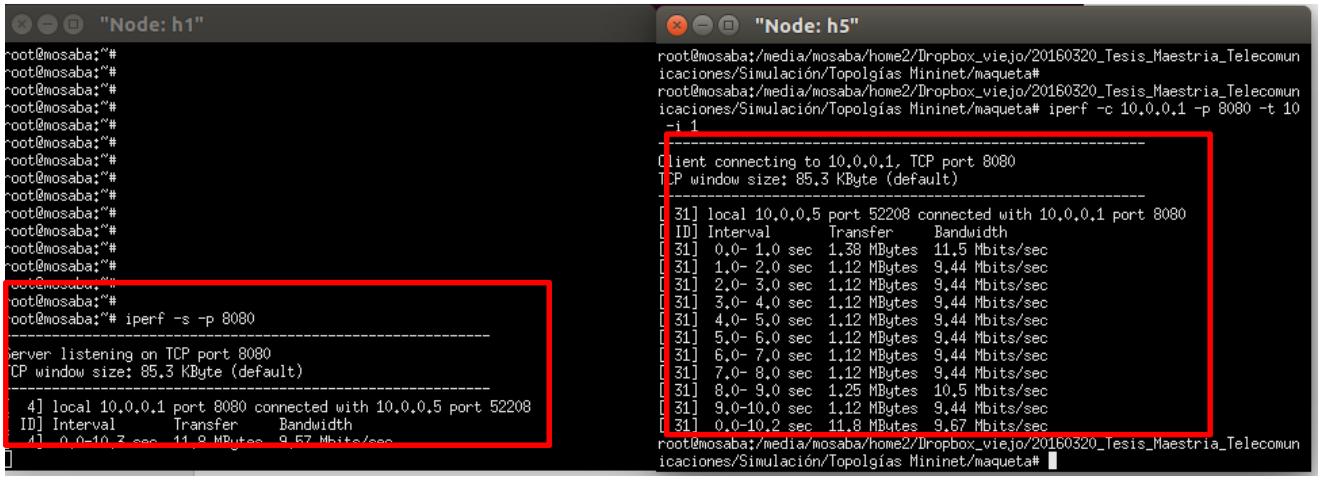


Figura Anexo 28 – Aislación de tráfico – Tráfico http entre h1 y h5

3) Se verifica la conectividad en el puerto 555 que controlan ambas redes virtuales (Figura Anexo 29).

Se configura en el host 1 iperf escuchando en el puerto 555:

```
# iperf -s –p 555
```

Se configura el host 5 para que envíe tráfico al host 1 en dicho puerto:

```
# iperf -c 10.0.0.1 -p 555 -t 10 -i 1
```

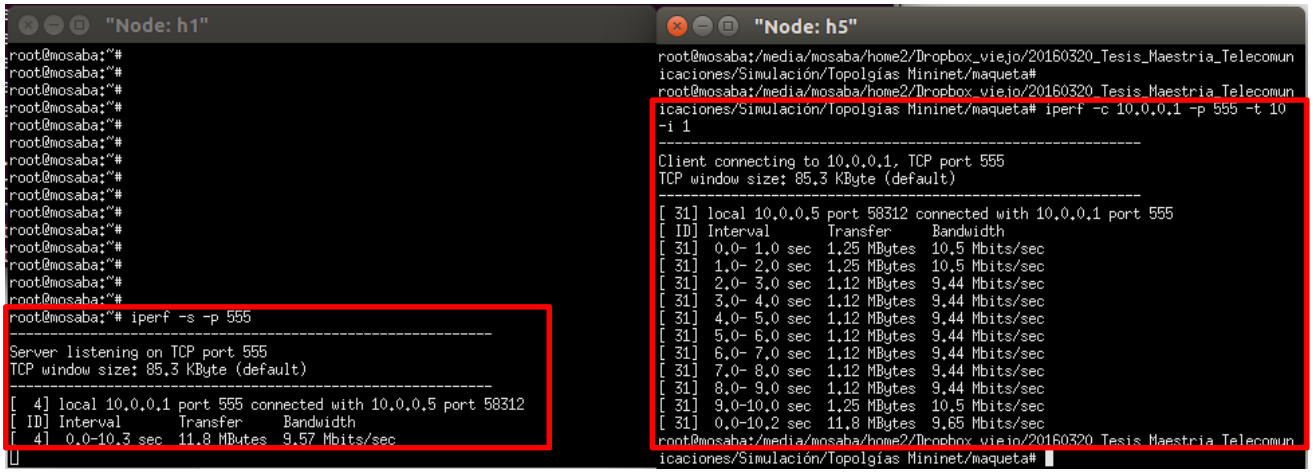


Figura Anexo 29 – Aislación de tráfico - Tráfico TCP puerto 555 entre h1 y h5

Luego se procede a bajar el servicio del controlador de la red naranja (Figura Anexo 30 y Figura Anexo 31).

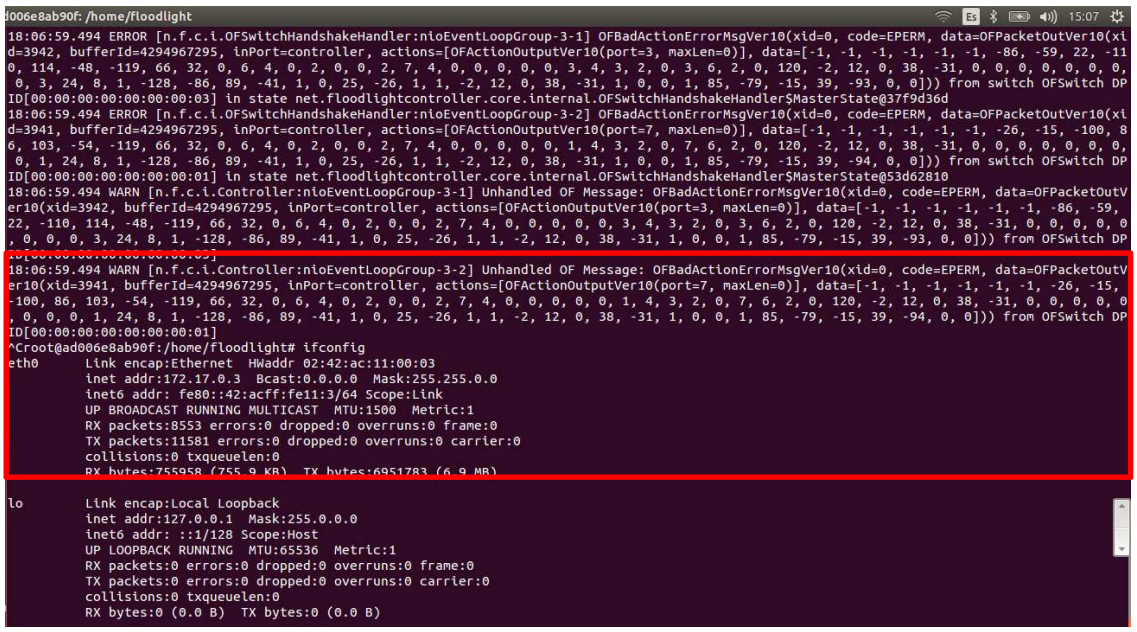


Figura Anexo 30 – Aislación de tráfico – Vista de la consola del controlador sin servicio

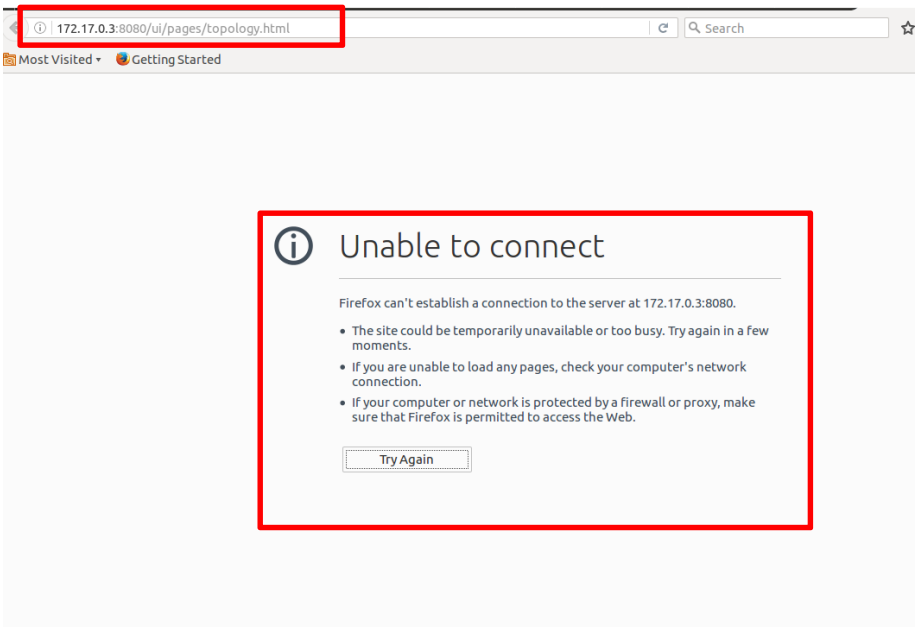


Figura Anexo 31- Aislación de tráfico – Vista de la interfaz web del controlador sin servicio

Se verifica que efectivamente que dicho controlador se encuentra fuera de servicio; luego, se realizan las mismas pruebas.

Se procede a evaluar la conectividad entre los hosts. Se debe tener en cuenta que al bajar el servicio del controlador "naranja", sólo queda en servicio el controlador "puerto_prueba". Este tiene capacidad de administrar únicamente los flujos correspondientes al puerto 555 por configuración a nivel de capa de virtualización (FlowVisor).

4) Realizamos un ping entre h1 y h5 (Figura Anexo 32).

```
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
^C
--- 10.0.0.5 ping statistics ---
19 packets transmitted, 0 received, 100% packet loss, time 18143ms
```

Figura Anexo 32- Aislación de tráfico – ping entre host 1 y 5

Tal como se puede apreciar, no existe respuesta en el ping entre estos dos hosts.

5) Se verifica la conectividad en el puerto 8080 (http) entre ambos (Figura Anexo 33).

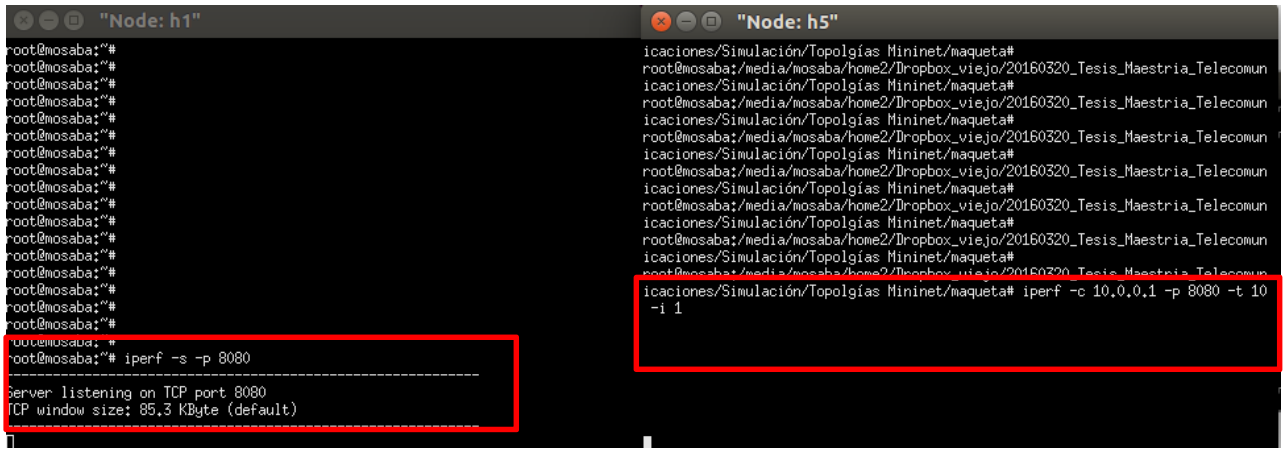


Figura Anexo 33- Aislación de tráfico – Tráfico http entre h1 y h5

Se aprecia que no se puede cursar tráfico a través de éste puerto.

6) Se verifica la conectividad en el puerto 555 que controlan ambas redes virtuales (Figura Anexo 34).

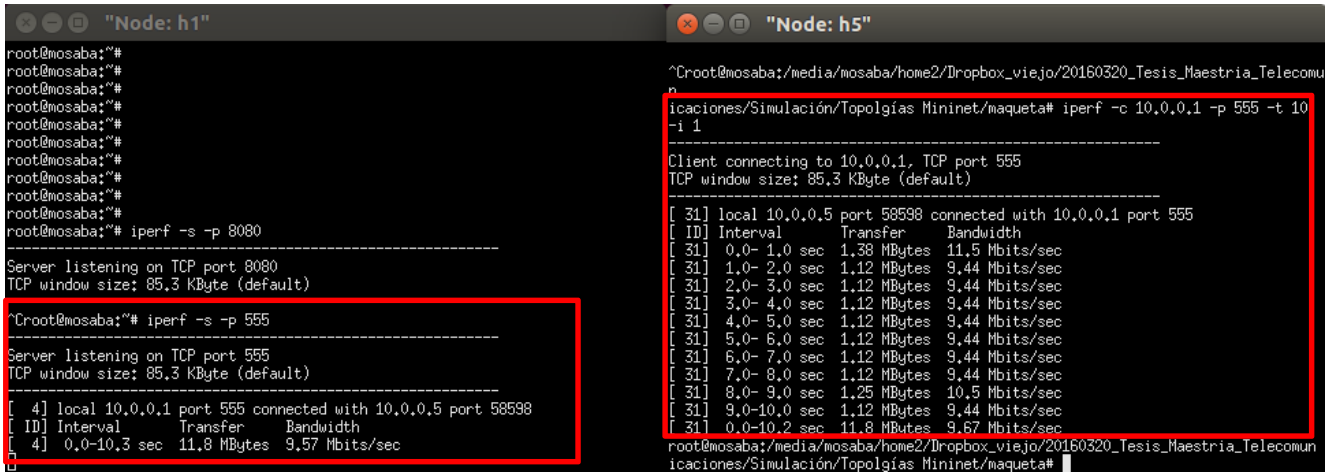


Figura Anexo 34 – Aislación de tráfico – Tráfico TCP puerto 555 entre host 1 y host 5

Tal como estaba previsto, a partir del diseño de las redes virtuales, efectivamente se pudo realizar una conexión entre el host 1 y 5 a través del puerto 555.

La topología visualizada desde el controlador de la red virtual puerto_prueba es la siguiente (Figura Anexo 35):

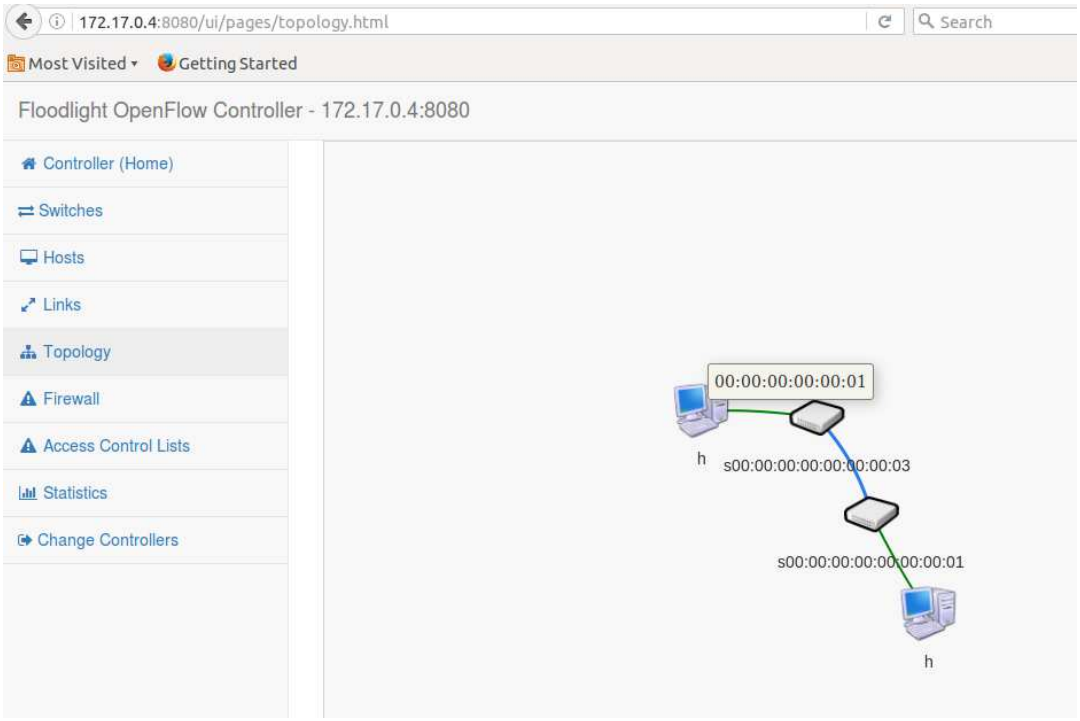


Figura Anexo 35- Aislación de tráfico – Topología visualizada desde controlador puerto_prueba

Esta simulación permite demostrar que él mismo "tipo" de tráfico puede ser compartido por dos o más controladores, quienes tendrán la capacidad de administración sobre el mismo.

10.ANEXO 10: APIS DE ADMINISTRACIÓN

i. *FlowVisor*

Ejecutando el siguiente comando en la consola del servidor donde se encuentra instalado FV, se obtiene el detalle de los diversos parámetros de administración que permite su API de administración:

```
# fvctl /dev/null --help
```

Usage: /usr/local/bin/fvctl [options] command [command_args]

Options:

-h HOST, --hostname=HOST
Specify the FlowVisor host; default='localhost'
-p PORT, --port=PORT Specify the FlowVisor web port; default=8081
-u FV_USER, --user=FV_USER
FlowVisor admin user; default='fvadmin'
-n, --no-passwd Run fvctl with no password; default false
-f FV_PASSWDFILE, --passwd-file=FV_PASSWDFILE
Password file; default=none
-v, --version
--help

Available commands are:

add-flowspace Creates a flowspace rule
add-slice Creates a new slice
get-config Displays the general purpose FlowVisor config params
list-datapath-flowdb Displays the contents of the flow db if flow tracking is enabled
list-datapath-flowrewritedb Displays the rewrites (or expansions) FlowVisor has applied
list-datapath-info Displays information for a connected device
list-datapath-stats Display statistics for a connected device
list-datapaths Displays the devices
list-flowspace Displays the flowspace
list-fs-status Check the insertion status of a add-flowspace or update-flowspace request.
list-fv-health Reports overall FlowVisor health
list-links Display overall topology
list-slice-health Reports overall slice health
list-slice-info Displays slice information
list-slice-stats Displays statistics about a slice

list-slices	Displays the configured slices
list-version	Displays FlowVisor version
register-event-callback	Registers your server, for events from FlowVisor
remove-flowspace	Remove a flowspace rule
remove-slice	Deletes a slice
save-config	Saves Flowvisor's configuration
set-config	Sets general purpose FlowVisor config parameters
unregister-event-callback	Unregisters your server from FlowVisor
update-admin-password	Update admin password
update-flowspace	Changes flowspace rule parameters
update-slice	Changes slice parameters
update-slice-password	Updates slice password

Nota: Los commandos expuestos corresponden a la versión 0.1 de la API.

ii. *OpenVirtex*

Ejecutando el siguiente comando en la consola del servidor donde se encuentra instalado OVX, se obtiene el detalle de los diversos parámetros de administración y monitoreo.

```
# python ovxctl.py -n --help
```

Usage: ovxctl.py [options] command [command_args]

Options:

-h HOST, --hostname=HOST
Specify the OpenVirteX host; default='localhost'
-p PORT, --port=PORT Specify the OpenVirteX web port; default=8080
-u OVX_USER, --user=OVX_USER

OpenVirtex admin user; default='admin'

-n, --no-passwd Run ovxctl with no password; default false

-v, --version

--help

Available commands are:

addControllers Adds controllers to a virtual switch

connectHost Connect host to a virtual port

connectLink Connect two virtual ports through a virtual link

connectRoute Connect two virtual ports inside a virtual big-switch

createNetwork Creates a virtual network

createPort Create virtual port

createSwitch Create virtual switch

disconnectHost Disconnect host from a virtual port

disconnectLink Disconnect link between two virtual ports

disconnectRoute Disconnect big-switch internal route between two virtual ports

getPhysicalFlowtable Get the physical flowtable of a specified switch or all switches

getPhysicalHosts Get a list of physical hosts

getPhysicalTopology Get the physical topology

getVirtualAddressMapping Get the virtual to physical address mapping for a specified virtual network

getVirtualFlowtable Get the flowtable in the specified virtual network

getVirtualHosts Get list of hosts in virtual network

getVirtualLinkMapping Get the virtual to physical link mapping

getVirtualSwitchMapping Get the virtual to physical switch mapping

getVirtualTopology Get the virtual topology

listVirtualNetworks Get a list of all virtual network tenant ID's

removeNetwork Remove a virtual network

removePort Remove virtual port

removeSwitch Remove virtual switch

setInternalRouting	Set big-switch internal routing mechanism
setLinkPath	Set the physical path of a virtual link
startNetwork	Start a virtual network
startPort	Start a virtual port
startSwitch	Start a virtual switch
stopNetwork	Stop a virtual network
stopPort	Shutdown a virtual port
stopSwitch	Shutdown a virtual switch

Nota. Los comandos expuestos corresponden a la versión 0.1 de la API.

11. ANEXO 11: SIMULACIÓN DE ATRIBUTOS: POLÍTICAS EXTENSIBLES Y RECONFIGURACIÓN EN CALIENTE

i. *OpenVirtex*

Se diseña la siguiente red virtual "celeste" (Figura Anexo 36).

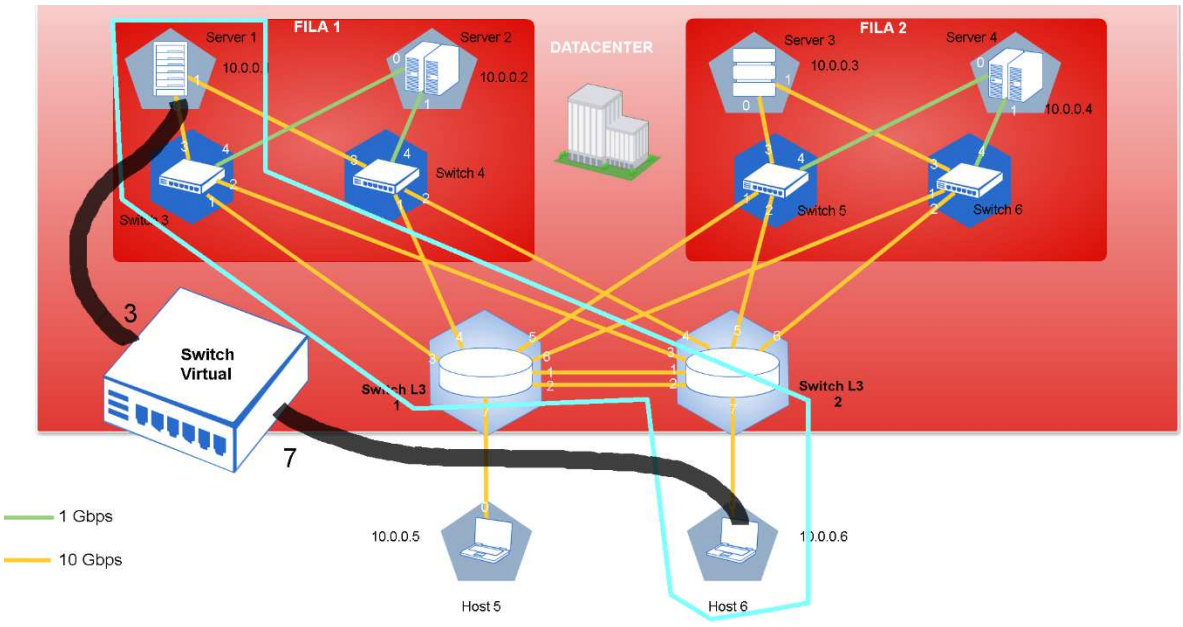


Figura Anexo 36 – Pol. Extensibles y Rec. en Caliente – Diseño de red celeste

El diseño de la red virtual que se muestra en la figura, se caracteriza por utilizar un switch virtual que permite la abstracción de los switches físicos 3 y L3 1, posibilitando

así, la conectividad entre el server/host 1 y el host 6. En contraposición, vemos que la red virtual “celeste” no permite la comunicación entre el server 1 y host 5 (Figura Anexo 37).

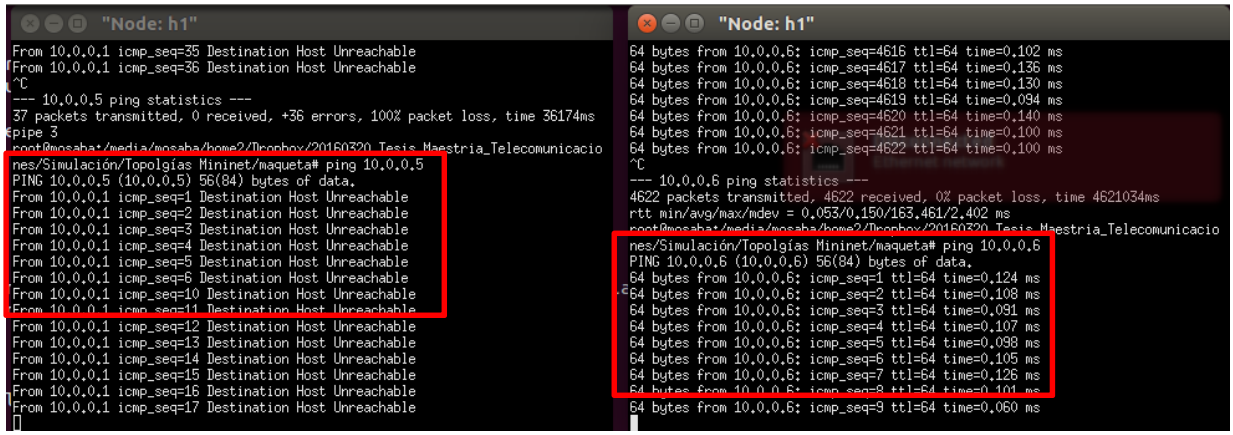


Figura Anexo 37 - Pol. Extensibles y Rec. en Caliente – ping desde server 1 a host 5 y host 6 respectivamente

Las experiencias propuestas a continuación, tiene por objetivo verificar la extensibilidad de las políticas que definen una red virtual, como así también la posibilidad de modificar las mismas en caliente.

La **primera experiencia**, consisten en modificar la configuración de la red virtual para que se pueda comunicar el host 1 con el host 5, manteniendo a su vez la comunicación con el host 6.

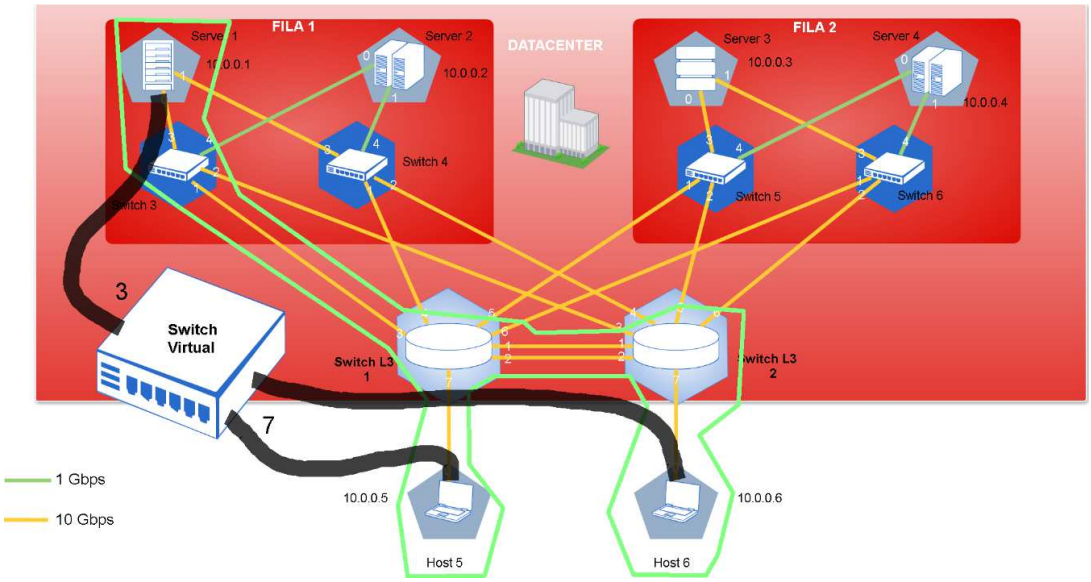


Figura Anexo 38 - Pol. Extensibles y Rec. en Caliente – extensión de la definición de la red para obtener conectividad con el host 5

Lo expuesto en la Figura Anexo 38, es posible a través de la definición del puerto virtual correspondiente y de la conexión del host 5 al switch virtual. Para ello, se deben ejecutar a través de la API de configuración de OVX, los siguientes comandos:

```
# python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 7
Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 3)
```

Creamos un puerto virtual correspondiente al puerto físico 7 del switch físico 1. Dicho puerto virtual se creo con el id = 3.

Luego se conecta el host 5 al puerto virtual 3 del switch virtual 00:a4:23:05:00:00:00:01 previamente creado:

```
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 3 00:00:00:00:00:05
Host (host_id 3) has been connected to virtual port
```

Una vez realizado ello, de forma inmediata, el host 1 comienza a tener comunicación con el host 5 tal como se puede apreciar en la imagen a continuación (Figura Anexo 39).

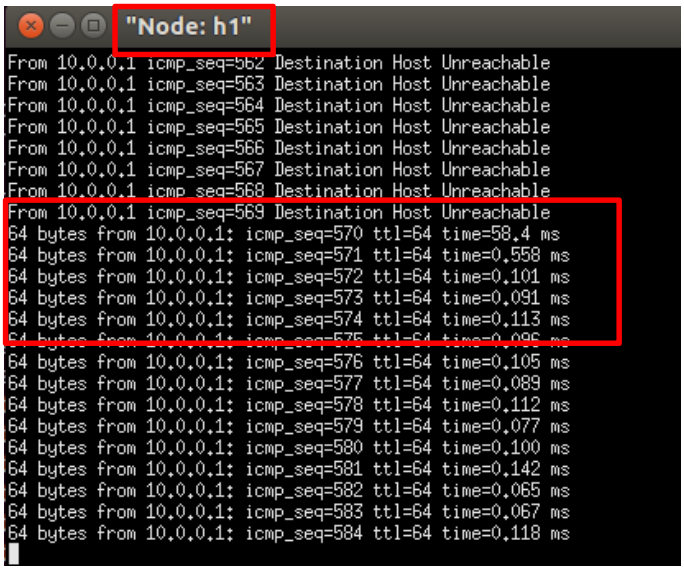


Figura Anexo 39 - Pol. Extensibles y Rec. en Caliente – ping h1 a h5 luego de modificar la configuración de la red

El cambio topológico de la red se puede visualizar desde la interfaz web del controlador (Figura Anexo 40).

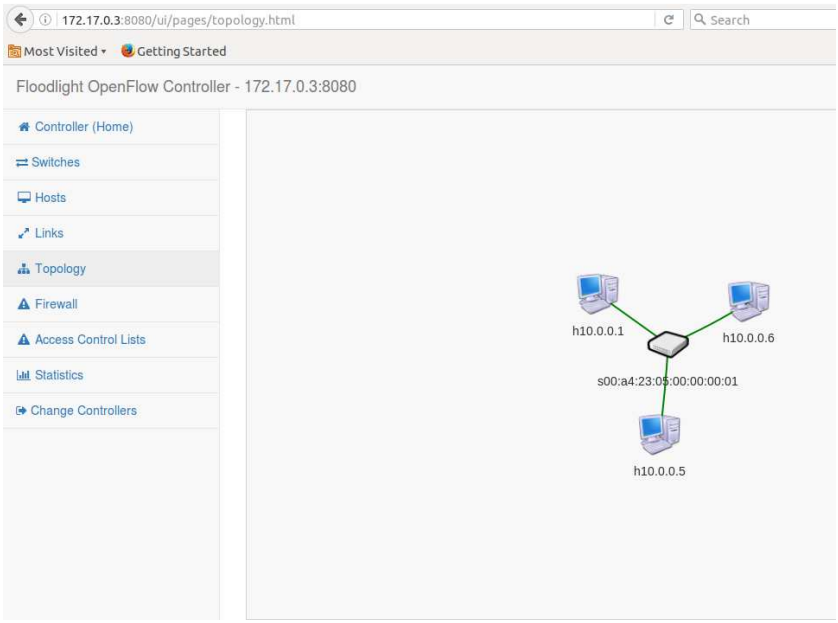


Figura Anexo 40 - Pol. Extensibles y Rec. en Caliente – Detalle de red virtual modificada. Se visualiza la conectividad entre los hosts 1, 6 y 5.

Como **segunda experiencia**, se modifica la red virtual para que el host 1 ya no se pueda comunicar con el host 6 configurando la red "verde" (Figura Anexo 41).

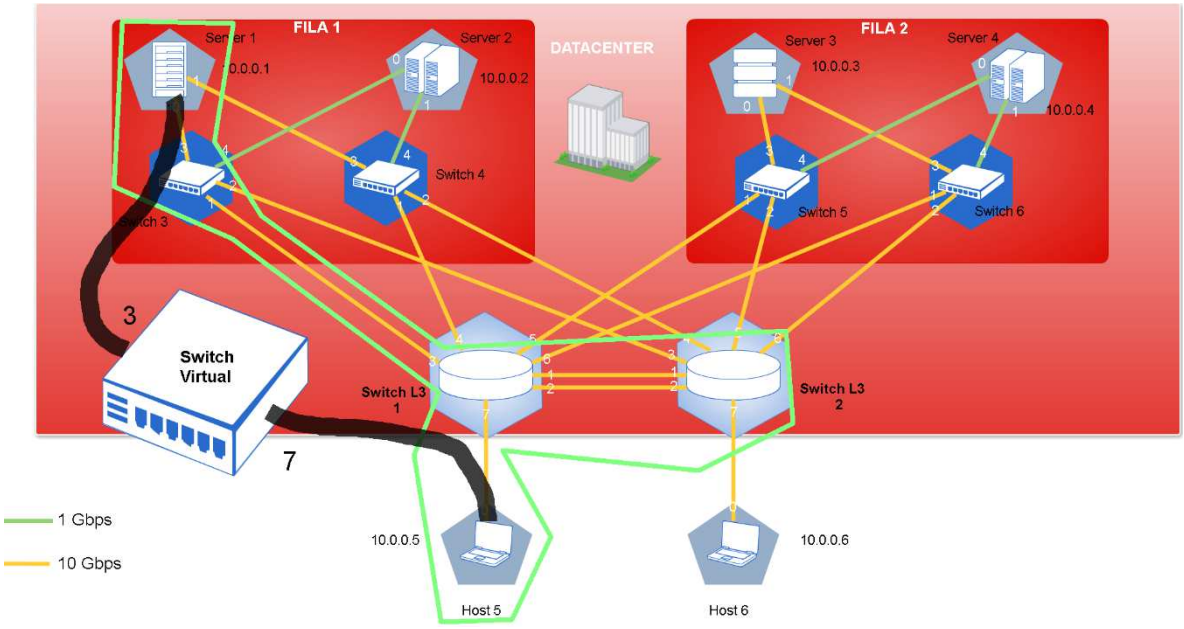


Figura Anexo 41 - Pol. Extensibles y Rec. en Caliente – Red verde

Para ello, se debe ejecutar el siguiente comando para remover el puerto correspondiente. Esto se realiza a través de la API de configuración de OVX.

python ovxctl.py -n removePort 1 00:a4:23:05:00:00:00:01 2

Port (port_id 2) has been removed from virtual switch (switch_id 00:a4:23:05:00:00:00:01)

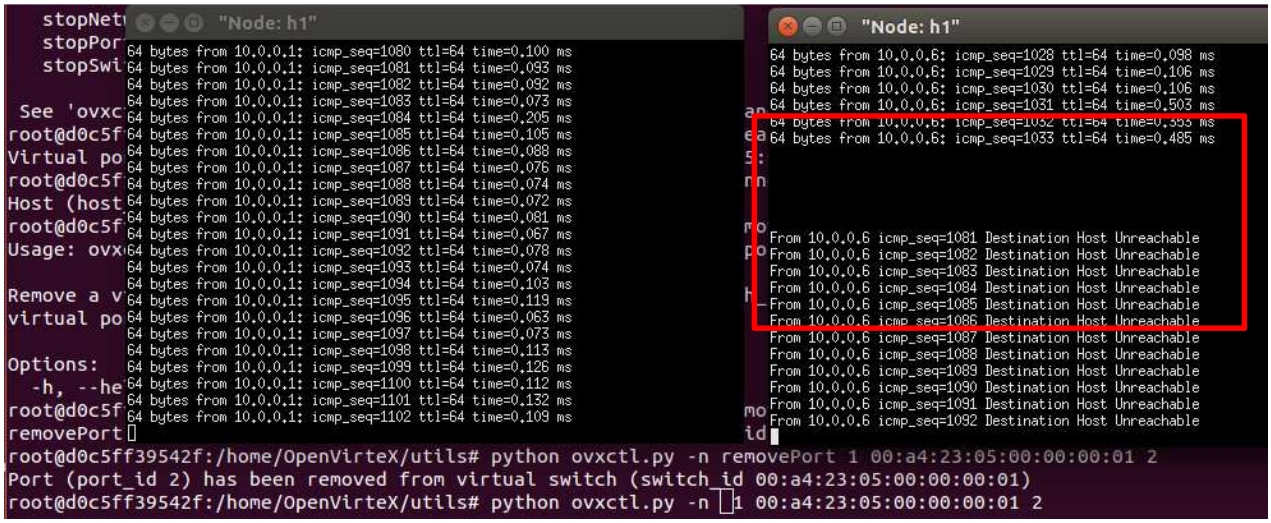


Figura Anexo 42 - Pol. Extensibles y Rec. en Caliente – Pérdida de conectividad entre h1 y h6.

De forma inmediata se pierde la conexión con el host 6 (Figura Anexo 42).

Ésta condición se puede visualizar también a través del controlador (Figura Anexo 43 y Figura Anexo 44).

Figura Anexo 43 - Pol. Extensibles y Rec. en Caliente – Pérdida de conectividad entre h1 y h6.

Hosts

Hosts Connected

MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen
00:00:00:00:00:01	10.0.0.1		00:a4:23:05:00:00:00:01	1	1466187203344
00:00:00:00:00:05	10.0.0.5		00:a4:23:05:00:00:00:01	3	1466187020187
00:00:00:00:00:06	10.0.0.6				1466186955515

Showing 1 to 3 of 3 entries

Figura Anexo 44 - Pol. Extensibles y Rec. en Caliente – Pérdida de conectividad entre h1 y h6.

Como **tercera experiencia**, se modificará la red virtual para poder tener comunicación con el host/servidor 3. La red virtual verde quedará entonces desde el punto de vista física y lógico de la siguiente manera:

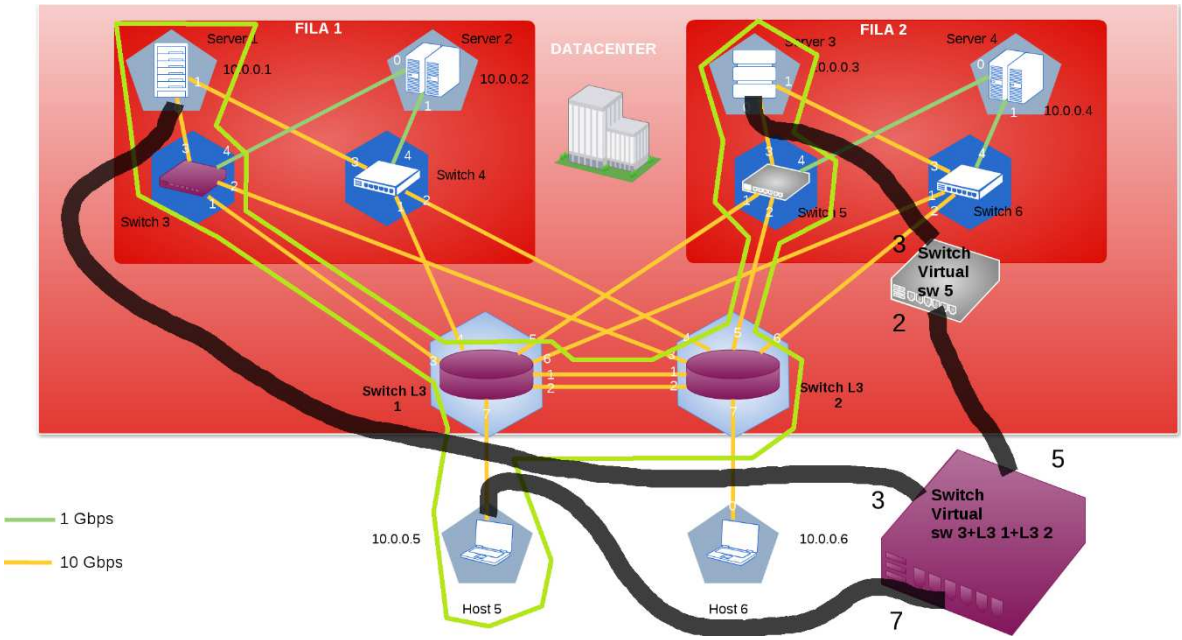


Figura Anexo 45 - Pol. Extensibles y Rec. en Caliente – extensión de la definición de la red para obtener conectividad con el server/host 3.

Para ello se ejecutan los siguientes comandos en la interfaz de administración de OVX:

1) Creación del switch virtual 2 comprendido por el switch físico 5

```
# python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:05
Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02)
```

2) Creación de los puertos virtuales correspondientes a los físicos 3 y 2 del switch 5.

```
# python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:05 3
Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 1)
# python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:05 2
Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 2)
```

3) Creación del puerto virtual correspondiente al puerto físico 5 del switch 2.

```
# python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:02 5
Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 5)
```

4) Conexión del switch virtual existente previamente y del switch virtual recientemente creado.

```
# python ovxctl.py -n connectLink 1 00:a4:23:05:00:00:00:01 5 00:a4:23:05:00:00:00:02
2 spf 1
Virtual link (link_id 1) has been created
```

5) Conexión del host/servidor físico 3 al switch virtual recientemente creado:

```
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:02 1 00:00:00:00:00:03
Host (host_id 5) has been connected to virtual port
```

Si se visualiza la topología de red vista por el controlador, se apreciará que efectivamante se consigue la incorporación de un switch virtual correspondiente al switch 5 de la red física y la conexión con el host/servidor 3 (Figura Anexo 46 y Figura Anexo 47).

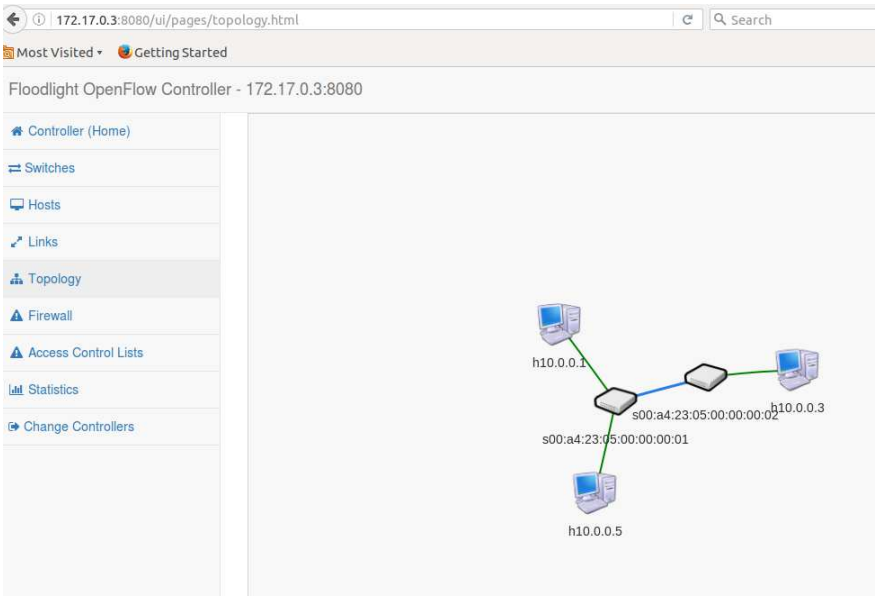


Figura Anexo 46 - Pol. Extensibles y Rec. en Caliente – extensión de la definición de la red para obtener conectividad con el server/host 3.

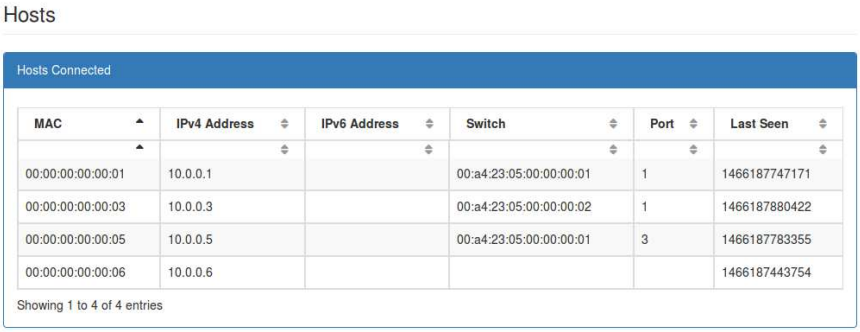


Figura Anexo 47 - Pol. Extensibles y Rec. en Caliente – extensión de la definición de la red para obtener conectividad con el server/host 3.

ii. *FlowVisor*

Se diseña la red celeste expuesta a continuación:

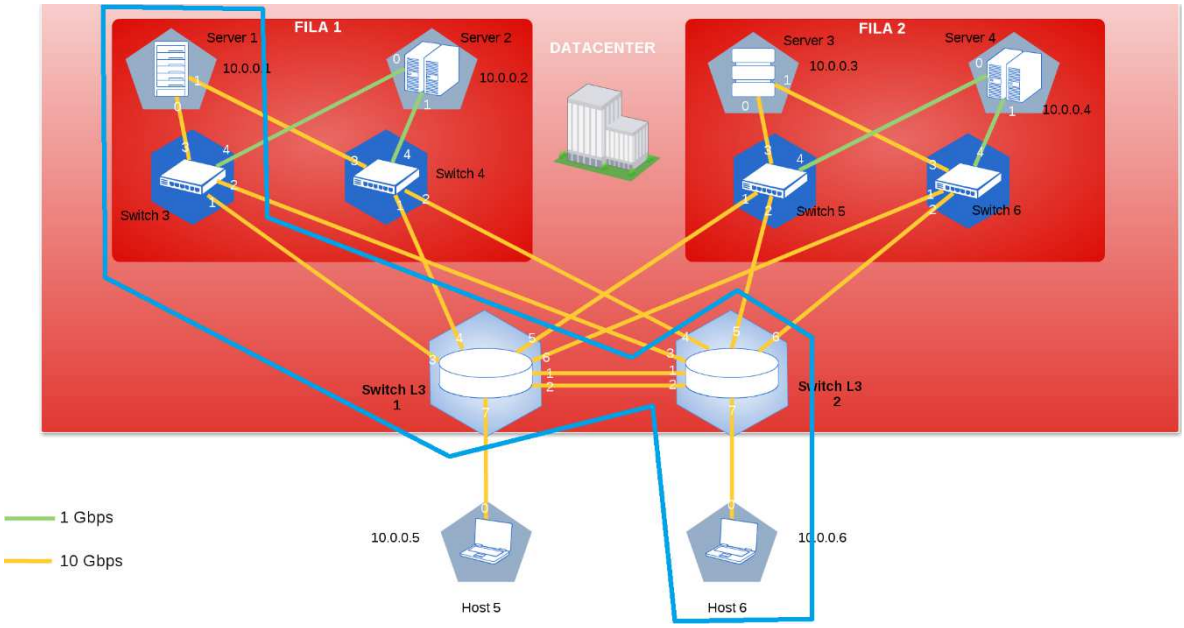


Figura Anexo 48 - Pol. Extensibles y Rec. en Caliente – Red Celeste

Tal como se puede observar, la misma es análoga a la red utilizada en OVX. La principal diferencia es que FV por su diseño, no permite la abstracción de la topología a través de un switch virtual. Sin embargo, al igual que lo realizado con OVX, se buscará la comunicación entre el host 1 y el host 6.

Se procede a configurar dicha red:

1) Creación del slice

fvctl -f /dev/null add-slice celeste tcp:172.17.0.3:10000 admin@celeste
Slice password:
Slice celeste was successfully created
Siendo 172.17.0.3 la IP y el puerto del controlador.

2) Creación/asignación de los espacios de flujo:

fvctl -f /dev/null add-flowspace dpid3-port3 3 1 in_port=3 celeste=7
#fvctl -f /dev/null add-flowspace dpid3-port1 3 1 in_port=1 celeste=7
FlowSpace dpid3-port1 was added with request id 1.
fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3 celeste=7
FlowSpace dpid1-port3 was added with request id 2.
fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1 celeste=7

FlowSpace dpid1-port1 was added with request id 3.

fvctl -f /dev/null add-flowspace dpid2-port1 2 1 in_port=1 celeste=7

FlowSpace dpid2-port1 was added with request id 4.

fvctl -f /dev/null add-flowspace dpid2-port7 2 1 in_port=7 celeste=7

FlowSpace dpid2-port7 was added with request id 5.

Luego se verifica que la red funcione adecuadamente a través de la ejecución de un ping desde h1 a h6 y viceversa, tal como se muestra en la imagen a continuación:

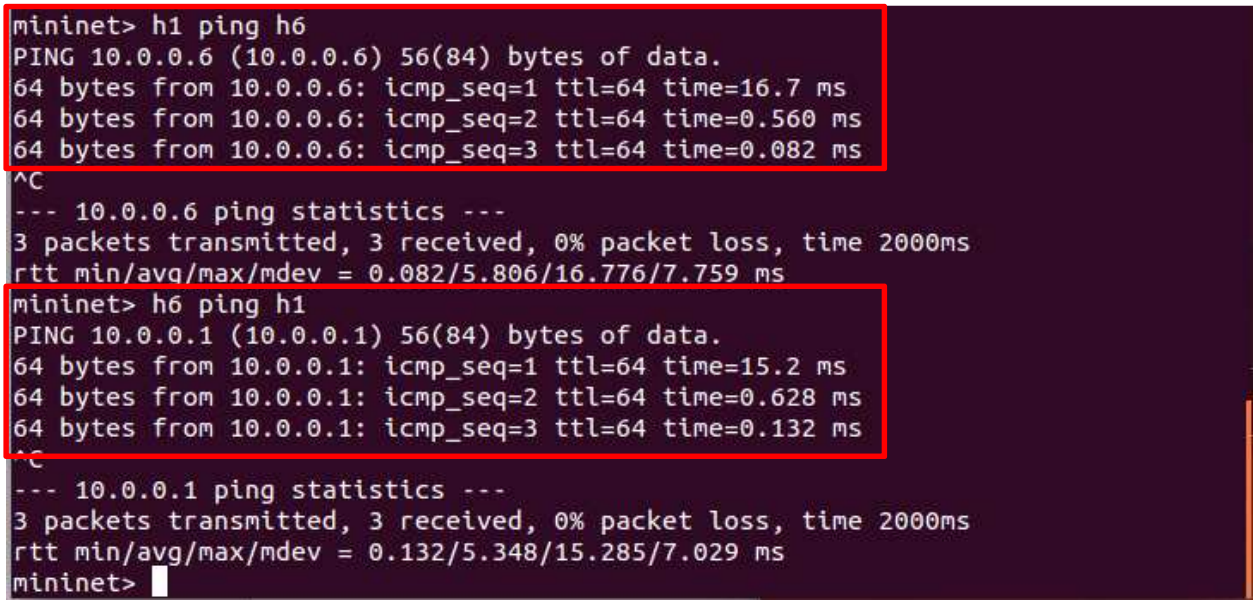


Figura Anexo 49 - Pol. Extensibles y Rec. en Caliente – ping desde server 1 a host 6

La topología de red que se visualiza a través de la interfaz del controlador es la que se presenta a continuación:

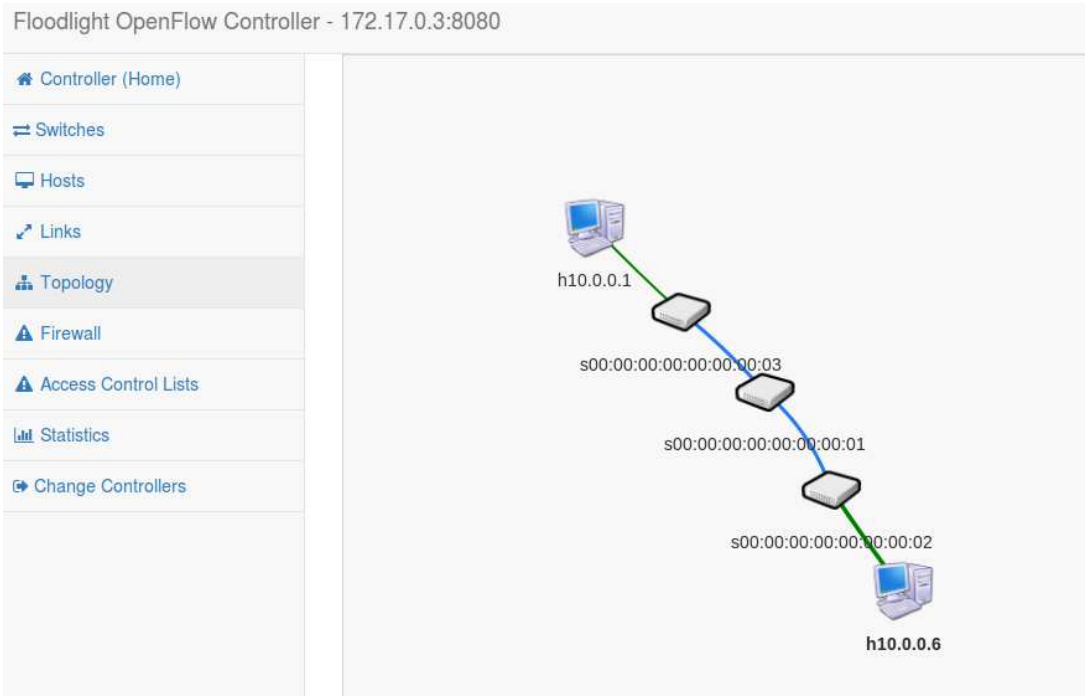


Figura Anexo 50 - Pol. Extensibles y Rec. en Caliente – Topología de red resultante vista desde el controlador (Figura Anexo 50)

Las siguientes experiencias tienen como objetivo verificar la extensibilidad de las políticas que definen una red virtual en FV y a su vez determinar si es posible realizar una modificación mientras la red está funcionando.

La **primera experiencia** es análoga a la presentada para OVX, es decir extender la política de la red para que sea posible la comunicación entre el host 1 y el 5 mientras que se mantiene la comunicación con el host 6.

Para ello se abren dos terminales del host 1 y se procede a realizar ping al host 5 y 6 de manera simultánea. La respuesta desde el host 6 es inmediata, mientras que, en forma previa a realizar dicha configuración, no hay respuesta desde el host 5 (Figura Anexo 51).

Luego se incorpora la siguiente definición desde la interfaz de administración de FV:

```
# fvctl -f /dev/null add-flowspace dpid1-port7 1 1 in_port=7 celeste=7
FlowSpace dpid1-port7 was added with request id 6.
```

Se logra la comunicación con el host 5 tal como se muestra en la imagen a continuación (Figura Anexo 51):

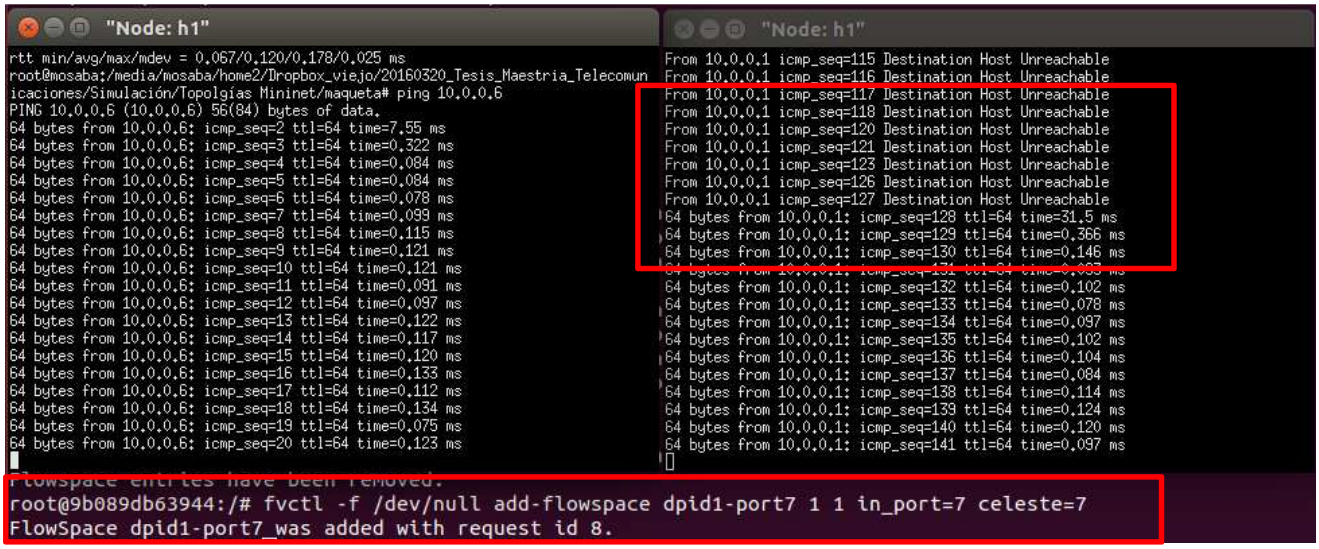


Figura Anexo 51 - Pol. Extensibles y Rec. en Caliente – ping desde server 1 a host host 6 y 5 respectivamente

La topología de red resultante que se visualiza desde el controlador es la siguiente (Figura Anexo 52):

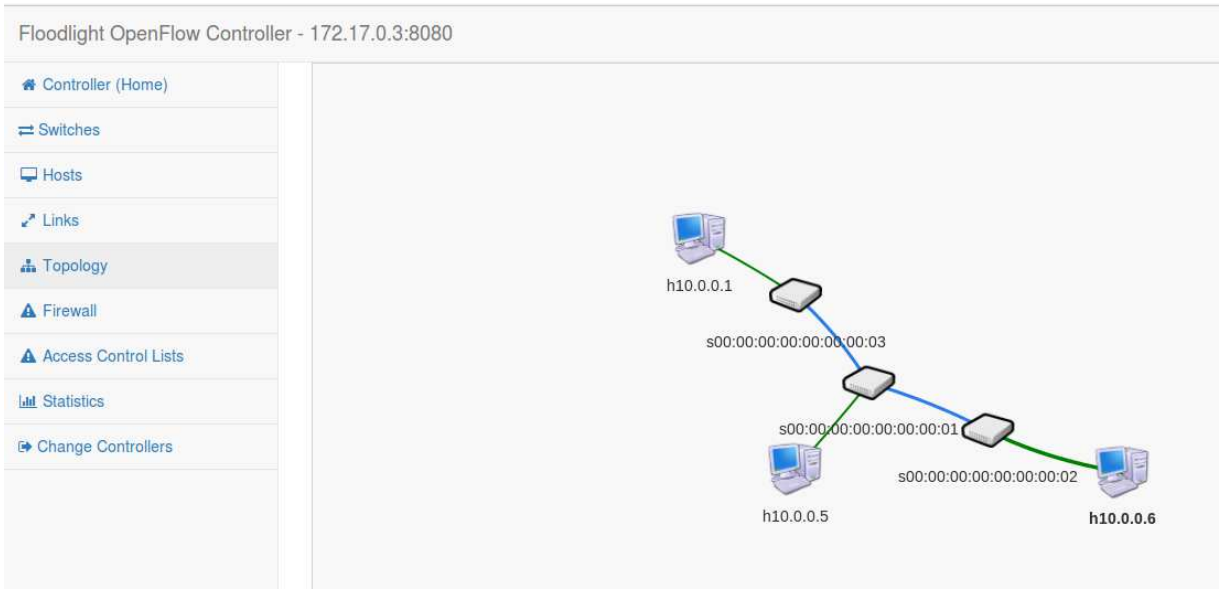


Figura Anexo 52 - Pol. Extensibles y Rec. en Caliente – Topología de red resultante

Como **segunda experiencia** se quitará la regla recién cargada para que la red quede configurada como la “red celeste” original (Figura Anexo 48):

fvctl -f /dev/null remove-flowspace dpid1-port7

Flowspace entries have been removed.

Luego de ello efectivamente el host1 pierde conectividad con el host 5.

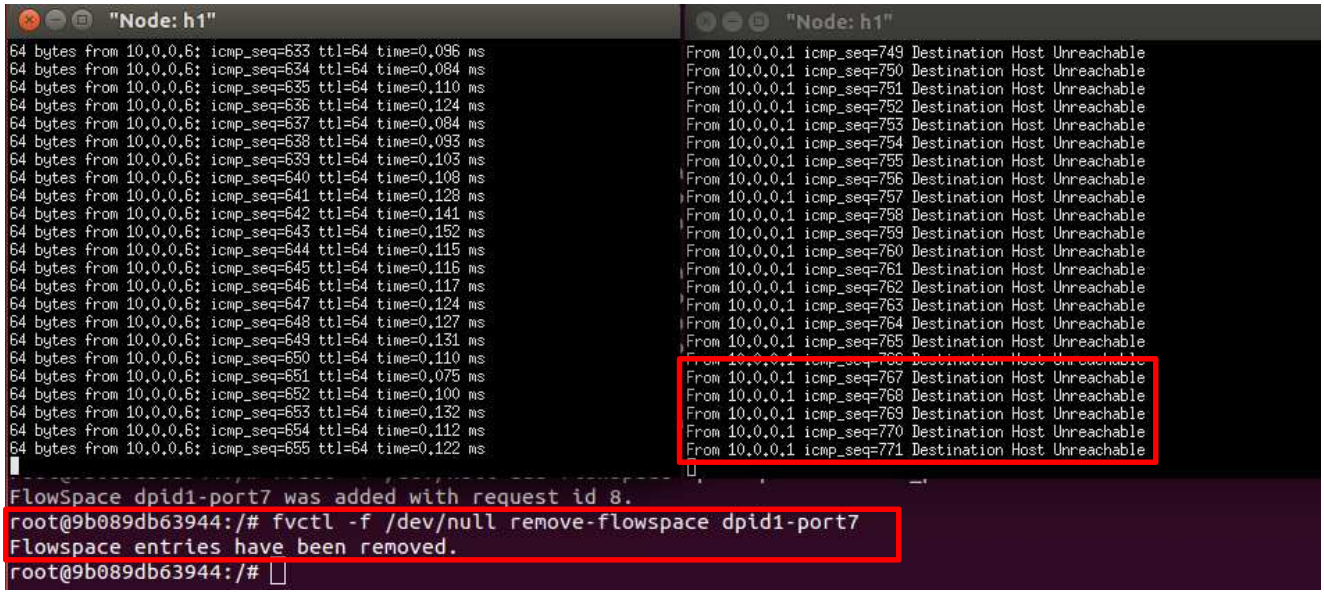


Figura Anexo 53 - Pol. Extensibles y Rec. en Caliente – Pérdida de conectividad entre h1 y h5

Como tercera experiencia, se extenderá la red virtual para tener conectividad con el host 3.

Se configuran las siguientes reglas:

fvctl -f /dev/null add-flowspace dpid2-port5 2 1 in_port=5 celeste=7

FlowSpace dpid2-port5 was added with request id 9.

fvctl -f /dev/null add-flowspace dpid5-port2 5 1 in_port=2 celeste=7

FlowSpace dpid5-port2 was added with request id 10.

fvctl -f /dev/null add-flowspace dpid5-port3 5 1 in_port=3 celeste=7

FlowSpace dpid5-port3 was added with request id 11.

Luego de ello, el posible la comunicación entre el host 1 y el 3. La topología de red virtual resultante se muestra a continuación (Figura Anexo 54):

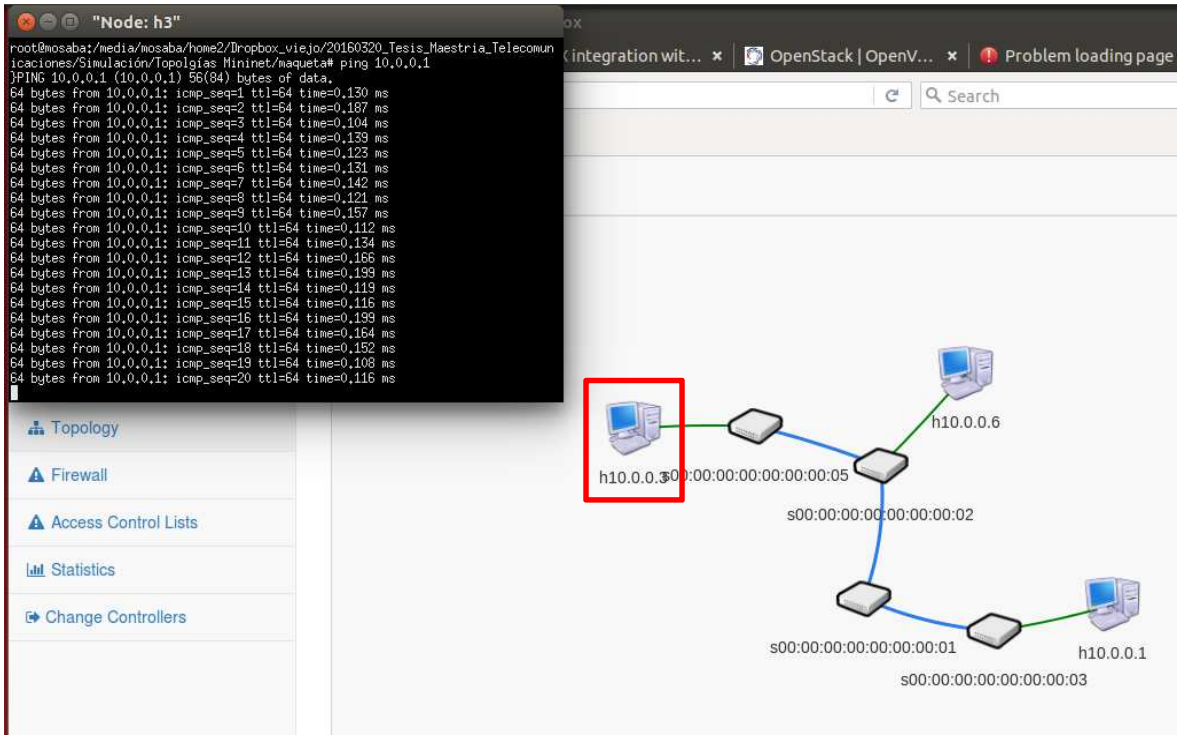


Figura Anexo 54 - Pol. Extensibles y Rec. en Caliente – Conectividad con el host 3

Tal como se puede apreciar, en la topología de red se visualiza el host 3.

Nota: Si bien fue posible, agregar y eliminar reglas mientras la red virtual presta servicios, se dieron oportunidades en las cuales las mismas no se actualizaron a nivel del controlador. Esta situación se dio en forma irregular, no pudiendo determinar las causas que lo originaron.

12. ANEXO 12: SIMULACIÓN DE ATRIBUTO DE ABSTRACCIÓN DE LA TOPOLOGÍA DE RED FÍSICA

i. *Abstracción de switches*

Se configura la red “naranja” que permite la conexión entre el host/server 1 y el 5 utilizando un switch virtual como abstracción del switch 3 y del switch L3 1 (Figura Anexo 55).

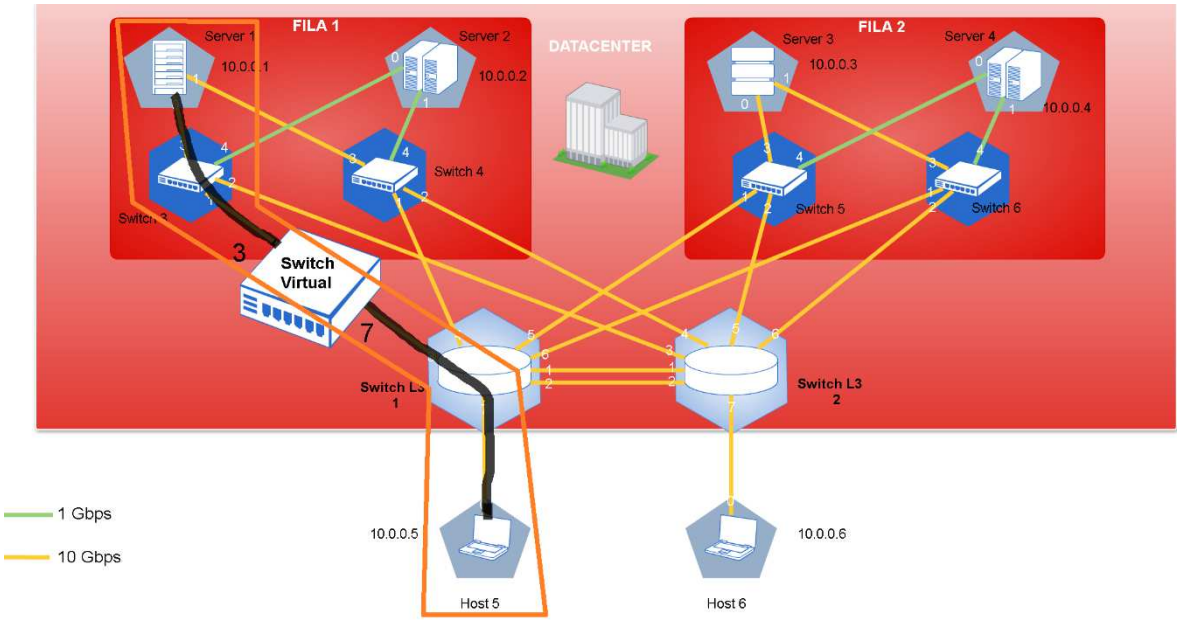


Figura Anexo 55- Abstracción de la topología de red – Diseño de red naranja utilizando un switch virtual

Se configura dicha red a partir de la consola de administración de OVX:

1) Creación de la red virtual:

python ovxctl.py -n createNetwork tcp:172.17.0.3:10000 10.0.0.0 16

2) Se crea un switch virtual como abstracción de dos switches físicos simulados.

#python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:03,00:00:00:00:00:00:00:01

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

3) Se crea el puerto virtual abstracción del puerto 3 correspondiente al switch 3

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 3

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

4) Se crea el puerto virtual abstracción del puerto 7 correspondiente al switch 1

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 7

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

5) Se conecta el host 1/servidor 1 al Puerto virtual 1

python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:01

Host (host_id 1) has been connected to virtual port

6) Se conecta el host 5 al puerto virtual 2

python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 2 00:00:00:00:00:05

Host (host_id 2) has been connected to virtual port

7) Se inicia la red

python ovxctl.py -n startNetwork 1Network (tenant_id 1) has been booted

Si se asigna el control de la red virtual a floodlight de acuerdo a lo expuesto en el punto 1), se visualiza la siguiente topología:

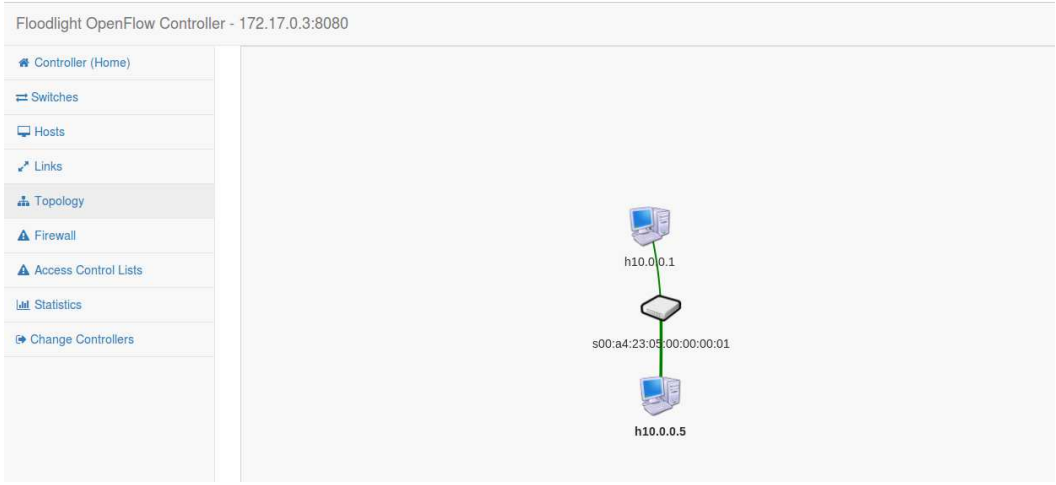


Figura Anexo 56 - Abstracción de la topología de red – Topología de red resultante. Observar que el switch virtual es una abstracción de los switches físicos de la red "naranja" (Figura Anexo 55)

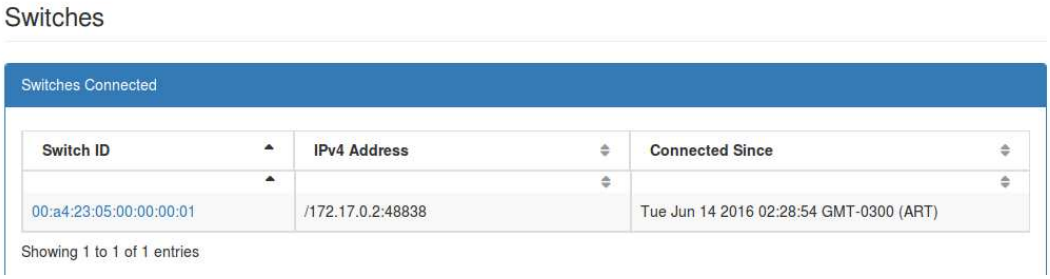


Figura Anexo 57 – Abstracción de la topología de red – Switch resultante como abstracción de los dos switches físicos 3 y L3 1 de la red "naranja".

Se demuestra que efectivamente el switch de la Figura Anexo 57 es una abstracción de los dos switches físicos de la red "naranja" (Figura Anexo 55) a través del siguiente comando, el cual se ejecuta en la consola de OVX con la siguiente respuesta:

```
# python ovxctl.py -n getVirtualSwitchMapping 1

{"00:a4:23:05:00:00:00:01": {"switches": ["00:00:00:00:00:00:00:03",
"00:00:00:00:00:00:00:01"], "links": [9, 7]}}
```

En la respuesta de dicho comando, se puede apreciar que el switch 00:a4:23:05:00:00:00:01 está compuesto por los switches 00:00:00:00:00:00:00:03, 00:00:00:00:00:00:00:01.

ii. *Abstracción de links*

Se configura la red "celeste", que permite la conexión entre el host/server 1 y el 5 utilizando un link virtual como abstracción de los links existentes entre el switch 3 y el switch L3 1 y de éste último, con respecto al switch L3 2 (Figura Anexo 58).

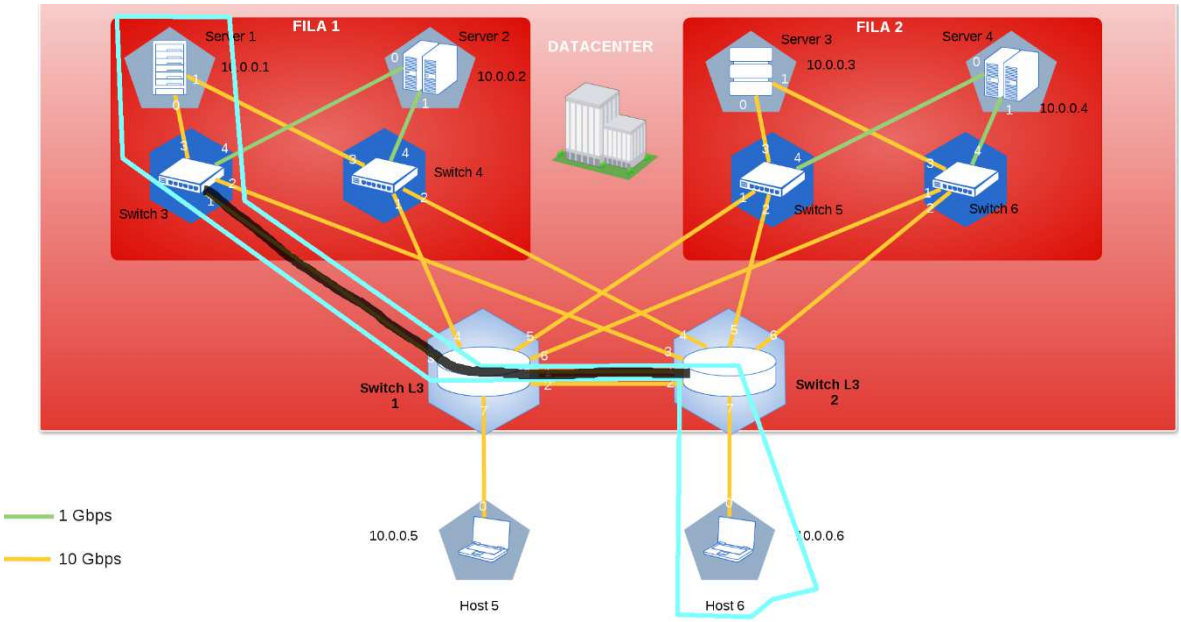


Figura Anexo 58 - Abstracción de la topología de red – Diseño de red celeste utilizando un link virtual

Dicha red se configura a través de la API de configuración de OVX:

1) Creación de la red virtual
   # python ovxctl.py -n createNetwork tcp:172.17.0.3:10000 10.0.0.0 16

2) Creación de los switches virtuales “de borde” a conectar:

python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:03

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:02

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:03)

Como se puede apreciar, solo se crean los switches virtuales que se “interconectan entre sí”, es decir el switch 3 y el switch L3 2.

3) Se crean los puertos virtuales requeridos en esos dos switches:

Switch 3:

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 3 Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

Switch L3 2:

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:02 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:03, port_id 1)

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:02 7

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:03, port_id 2)

4) Interconexión entre los puertos de los switches a través de los puertos virtuales creados (puerto físico 3 y 1 respectivamente).

python ovxctl.py -n connectLink 1 00:a4:23:05:00:00:00:01 2 00:a4:23:05:00:00:00:03 1 spf 1

5) Conexión de los Host 1/Servidor 1 y el host 6:

python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:01

Host (host_id 1) has been connected to virtual port

python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:03 2 00:00:00:00:00:06

Host (host_id 2) has been connected to virtual port

6) Inicio de la red

python ovxctl.py -n startNetwork 1

Network (tenant_id 1) has been booted

Si se visualiza la topología de red de acuerdo al controlador asignado en 1), se podrá apreciar que es posible enviar tráfico entre el server1/host1 y el host 6 a través de un “link” virtual entre los switches de borde de la red. Esta condición de diseño resulta transparente al usuario, tal como se puede apreciar en la topología de red vista por el controlador (Figura Anexo 59).

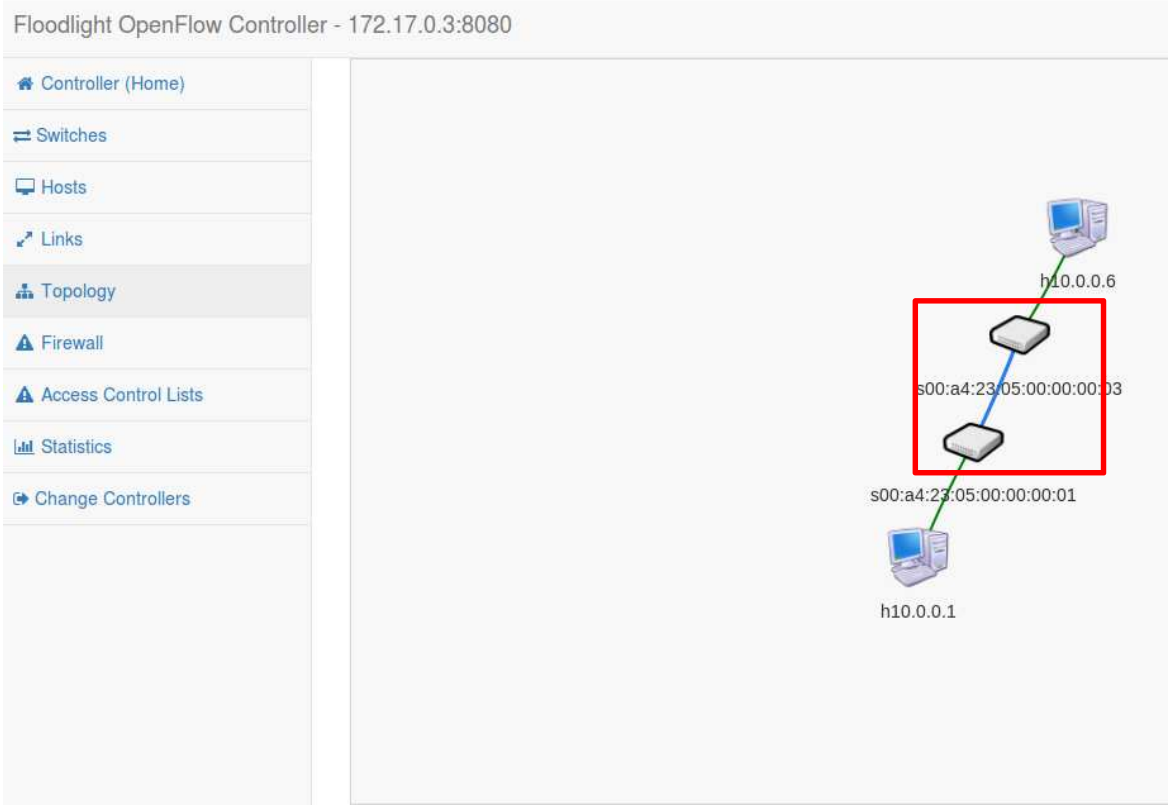


Figura Anexo 59 - Abstracción de la topología de red – Topología de red resultante

Se procede a realizar una prueba de ping entre el host/server 1 y host 6 (Figura Anexo 60).

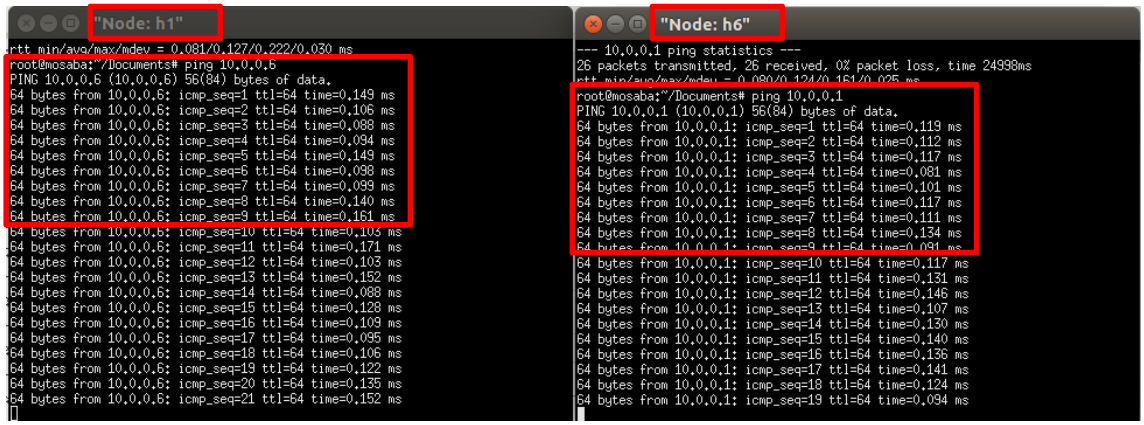


Figura Anexo 60 - Abstracción de la topología de red – Ping entre el host 6 y el host/server 1

13. ANEXO 13: SIMULACIÓN DE ATRIBUTO DE ALTA DISPONIBILIDAD/RESILIENCIA

Se configura la red virtual “celeste” según se detalla en la Figura Anexo 61.

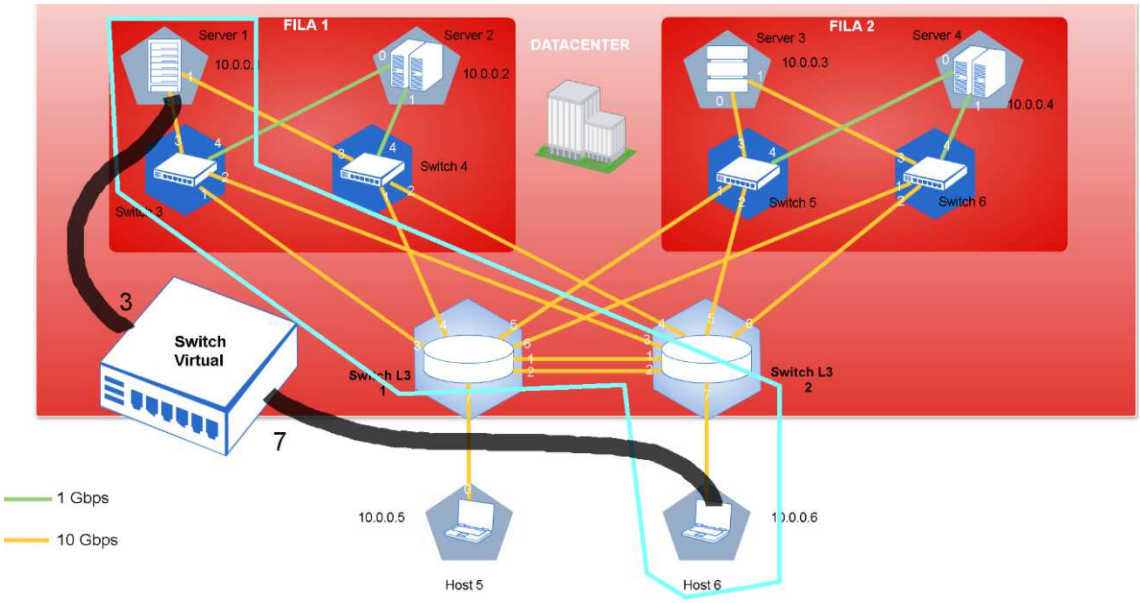


Figura Anexo 61 – Alta disponibilidad/Resiliencia – Red celeste

La configuración de la misma se detalla a continuación:

1) Se crea de la red virtual y se asigna al controlador Floodlight

   # python ovxctl.py -n createNetwork tcp:172.17.0.3:10000 10.0.0.0 16
   Virtual network has been created (network_id {u'mask': 16, u'networkAddress': 167772160, u'controllerUrls': [u'tcp:172.17.0.3:10000'], u'tenantId': 1}).

2) Generación de un switch virtual compuesto por los switches 3 L3 1 y L3 2.

   # python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:03,00:00:00:00:00:00:00:01,00:00:00:00:00:00:00:02
   Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

3) Creación de los puertos virtuales correspondientes a los puertos físicos 3 del switch 3 y el 7 del switch L3 2.

   # python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:03 3
   Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)
   # python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:02 7

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

4) Conexión del host/servidor 1 y 6 a los puertos virtuales correspondientes

```
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:01
Host (host_id 1) has been connected to virtual port
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 2 00:00:00:00:00:06
Host (host_id 2) has been connected to virtual port
```

5) Inicio de la red virtual

```
# python ovxctl.py -n startNetwork 1
Network (tenant_id 1) has been booted
```

La topología de red generada se aprecia en la Figura Anexo 62 - Alta disponibilidad/Resiliencia – Red celeste vista desde el controladorfigura a continuación:

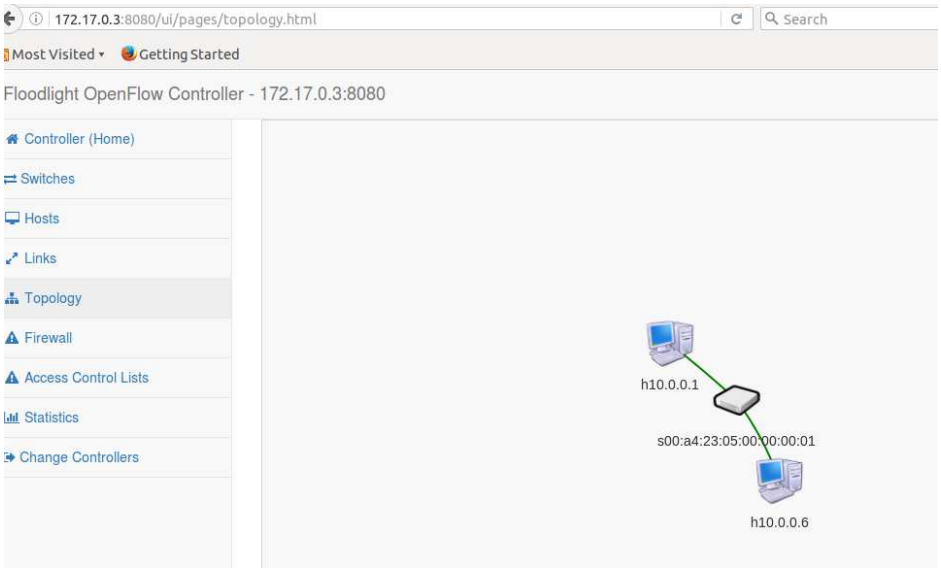


Figura Anexo 62 - Alta disponibilidad/Resiliencia – Red celeste vista desde el controlador

Se procede a realizar ping desde la termina del host 1 hacia el host 6 (Figura Anexo 63).

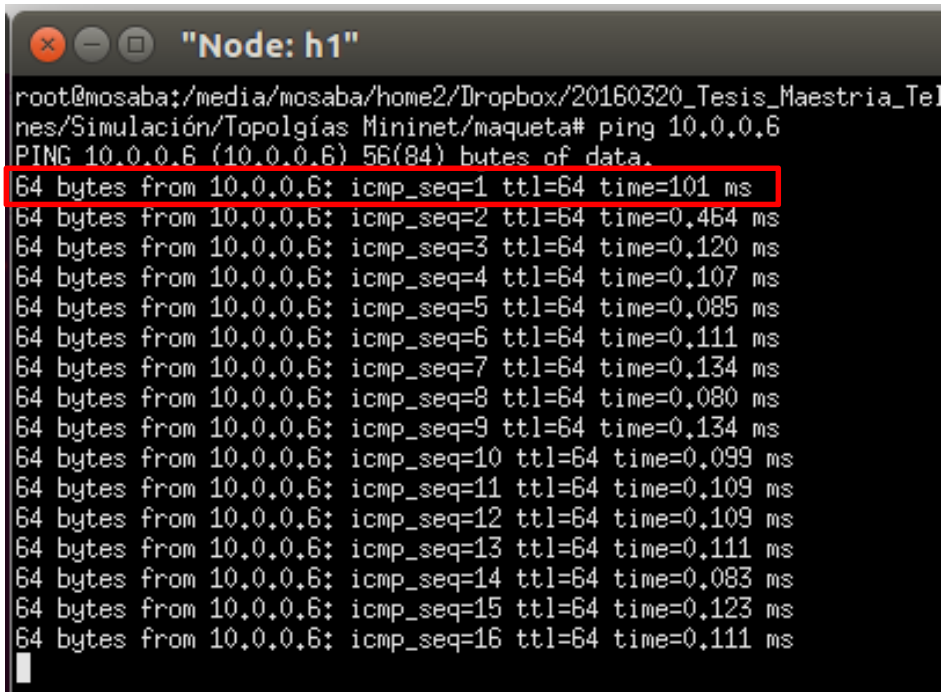


Figura Anexo 63 - Alta disponibilidad/Resiliencia – Ping host 1 al host 6

Se puede apreciar que, debido al proceso de carga de las reglas de flujo, el primer ping tiene un tiempo de respuesta sensiblemente superior al del resto.

Tal como se aprecia en la Figura Anexo 61, independientemente de que OVX genere una abstracción de los tres switches a través de uno virtual, existen dos caminos de comunicación entre los host/servidores 1 y 6.

El primer camino es a través del puerto 2 del switch 3, mientras que el segundo camino es a través de su puerto 1 pasando por el switch L3 1.

Si se analizan las tablas de flujo de los switches L3 2 y S3 respectivamente a través de mininet, veremos que por defecto el tráfico toma el camino más corto:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x100000003, duration=387.407s, table=0, n_packets=386,
n_bytes=37828, idle_timeout=5, idle_age=1,
priority=1,ip,in_port=7,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01,nw
```

```
_src=10.0.0.6,nw_dst=10.0.0.1
actions=mod_nw_dst:1.0.0.1,mod_nw_src:1.0.0.2,output:3
 cookie=0x100000002, duration=387.423s, table=0, n_packets=386,
n_bytes=37828, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06,nw
_src=1.0.0.1,nw_dst=1.0.0.2
actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.6,output:7
*** s3 ------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x100000002, duration=387.424s, table=0, n_packets=386,
n_bytes=37828, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06,nw
_src=10.0.0.1,nw_dst=10.0.0.6
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:2
 cookie=0x100000003, duration=387.421s, table=0, n_packets=386,
n_bytes=37828, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=2,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01,nw
_src=1.0.0.2,nw_dst=1.0.0.1
actions=mod_nw_src:10.0.0.6,mod_nw_dst:10.0.0.1,output:3
*** s4 ------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s5 ------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s6 ------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
```

Adicionalmente y para verificar este comportamiento, se procede a analizar el tráfico en el puerto 2 del switch 3 a través de wireshak. Se coloca un filtro para que sólo se muestre el tráfico ICMP (Internet Control Message Protocol) (Figura Anexo 64).

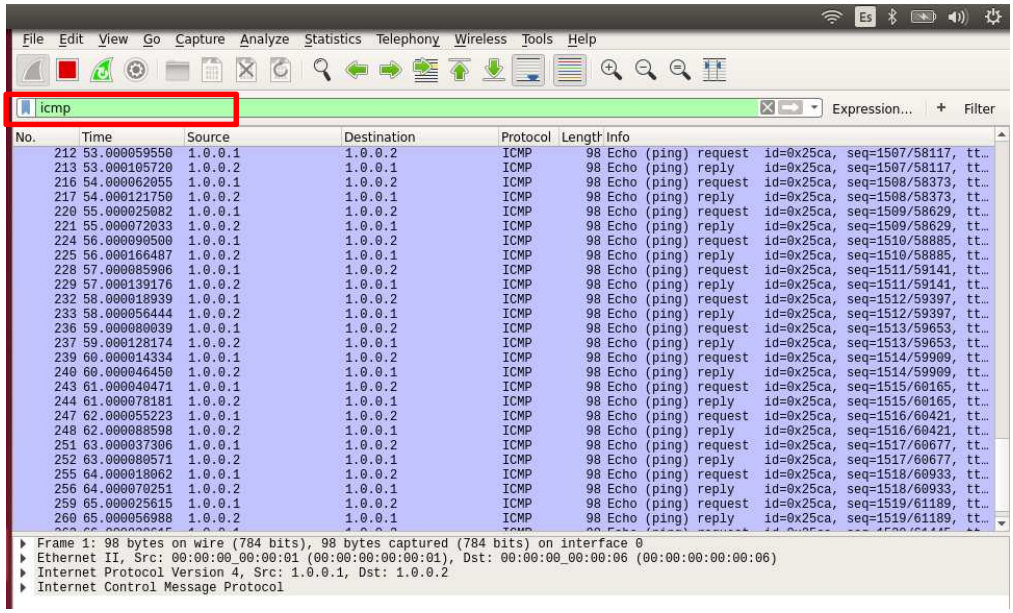


Figura Anexo 64 - Alta disponibilidad/Resiliencia – Tráfico en el puerto 2 del switch 3.

El tráfico ICMP efectivamente se ve a través del puerto 2 del switch 1.

Aquí se debe observar, que las direcciones IP no son la de origen y destino (10.0.0.1 y 10.0.0.6), ya que como se aprecia en la tabla de flujo de éste switch, el mismo realiza un proceso de conversión. Esto se debe a que OVX permite utilizar en una misma red física el mismo segmento de red para distintos usuarios, tal como se aprecia en ANEXO 8: Simulación de atributo de aislación.

Luego se procede a ejecutar el siguiente comando para indicarle a OVX que cree caminos alternativos, en caso de que un link físico falle:

```
# python ovxctl.py -n setInternalRouting 1 00:a4:23:05:00:00:00:01 spf 2
```

En el log de OVX se verá el siguiente mensaje:

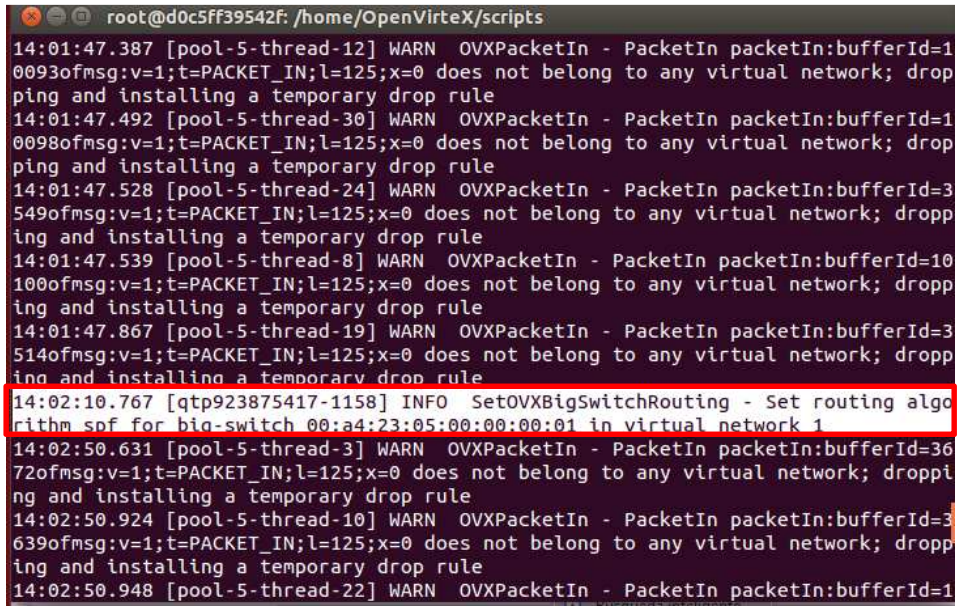


Figura Anexo 65 - Alta disponibilidad/Resiliencia – Log de OVX luego de la creación de caminos alternativos

Para probar el atributo de resiliencia de OVX, se ejecuta el siguiente comando en mininet, cuyo objetivo es bajar el servicio del link físico entre los switches 3 y L3 switch 2.

mininet> link s3 s2 down

Este comando pone en estado “bajo” el camino entre los switches 1 y 2, que representaba el vínculo sobre el cual se estaba traficando ICMP entre los dos hosts.

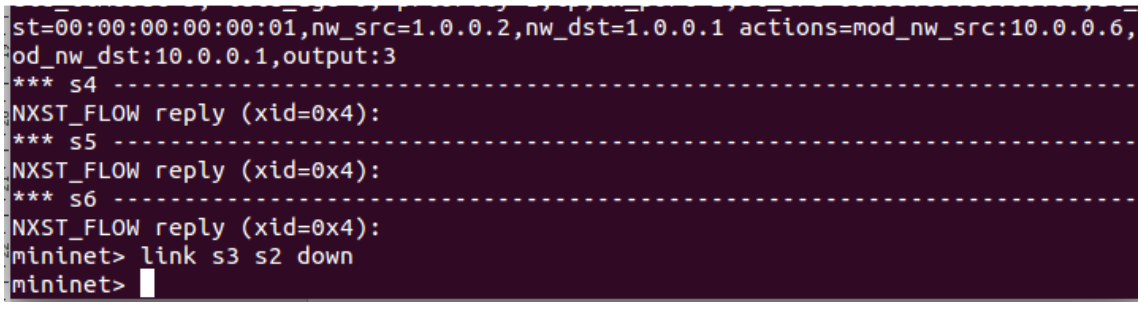


Figura Anexo 66 - Alta disponibilidad/Resiliencia – Baja del servicio del link entre los switches

Se procede a verificar que el servicio en dicho link se dio efectivamente de baja a través de Wireshark (Figura Anexo 67).

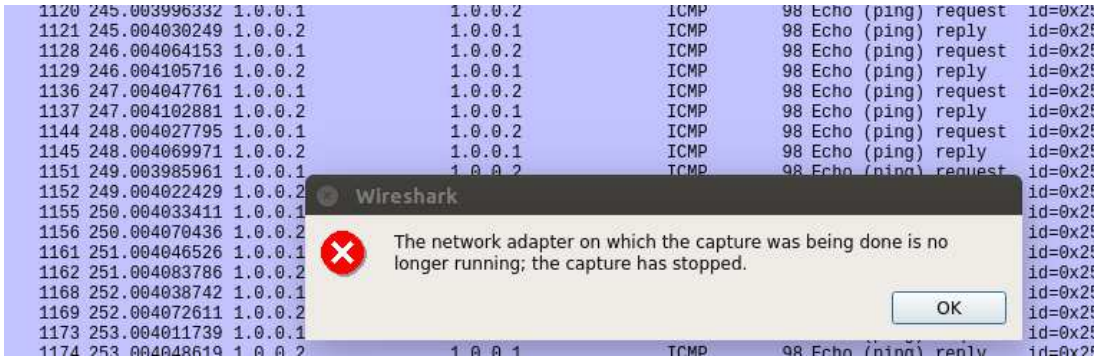


Figura Anexo 67 - Alta disponibilidad/Resiliencia – Baja del servicio del link entre los switches visto desde Wireshark

Para verificar la resiliencia de la solución, se procede a visualizar la terminal del host 1, el cual se encontraba realizando ping contra el host 6. Se puede apreciar que luego de un instante el vínculo se recupera (ello se hace notorio ya que como vemos el tiempo de respuesta en uno de los *pings* es mayor).

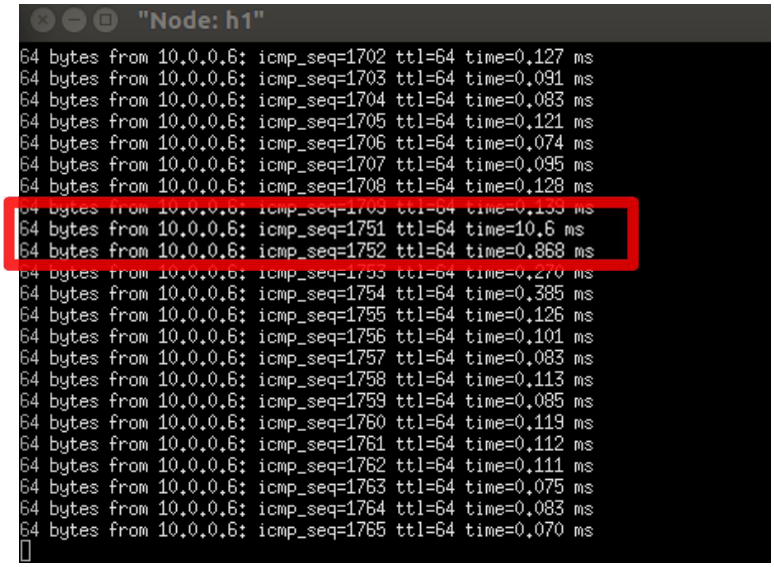


Figura Anexo 68 - Alta disponibilidad/Resiliencia – Comunicación entre el h1 y el h6 luego de que se da de baja el servicio del enlace principal

Este tiempo de “demora” se da ya que el controlador al detectar la “falla”, debe actualizar las tablas de flujo.

Luego de ejecutar el siguiente comando en mininet, podremos apreciar ello:

mininet> dpctl dump-flows

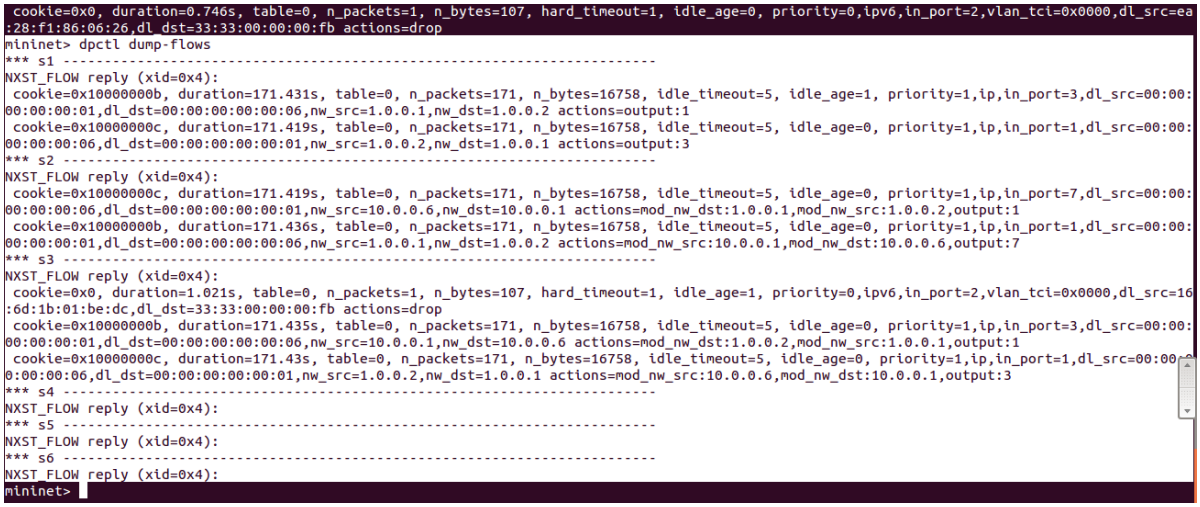


Figura Anexo 69 - Alta disponibilidad/Resiliencia – Tablas de flujo de los switches de la red luego de que se da de baja el servicio del enlace principal

Se presentan de manera más clara a continuación:

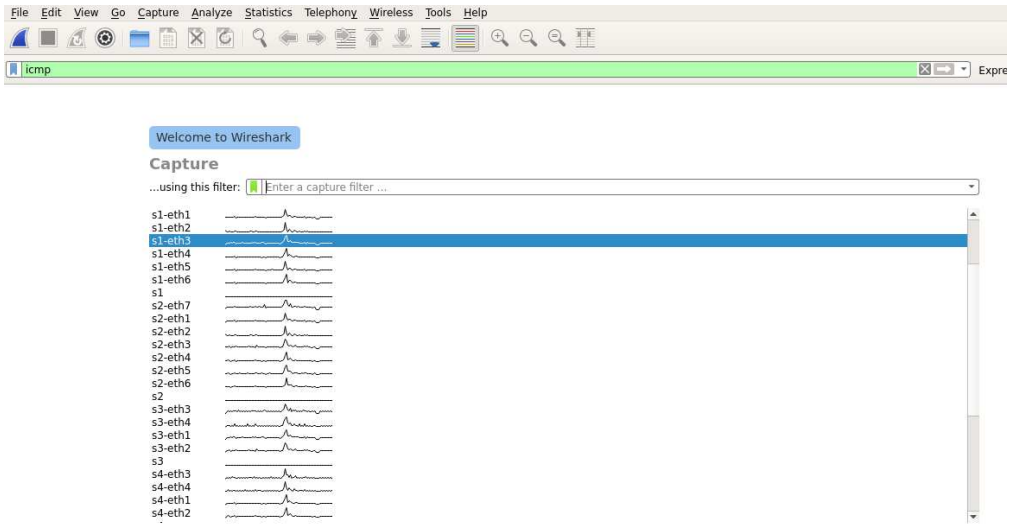
mininet> dpctl dump-flows
*** s1 ----------
NXST_FLOW reply (xid=0x4):
cookie=0x10000000b, duration=171.431s, table=0, n_packets=171, n_bytes=16758, idle_timeout=5, idle_age=1, priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06,nw_src=1.0.0.1,nw_dst=1.0.0.2 actions=output:1
cookie=0x10000000c, duration=171.419s, table=0, n_packets=171, n_bytes=16758, idle_timeout=5, idle_age=0, priority=1,ip,in_port=1,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01,nw_src=1.0.0.2,nw_dst=1.0.0.1 actions=output:3
*** s2 ----------
NXST_FLOW reply (xid=0x4):
cookie=0x10000000c, duration=171.419s, table=0, n_packets=171, n_bytes=16758, idle_timeout=5, idle_age=0, priority=1,ip,in_port=7,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.6,nw_dst=10.0.0.1
actions=mod_nw_dst:1.0.0.1,mod_nw_src:1.0.0.2,output:1

```
 cookie=0x10000000b, duration=171.436s, table=0, n_packets=171,
n_bytes=16758, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06,nw
_src=1.0.0.1,nw_dst=1.0.0.2
actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.6,output:7
*** s3 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1.021s, table=0, n_packets=1, n_bytes=107,
hard_timeout=1, idle_age=1,
priority=0,ipv6,in_port=2,vlan_tci=0x0000,dl_src=16:6d:1b:01:be:dc,dl_dst=33:
33:00:00:00:fb actions=drop
 cookie=0x10000000b, duration=171.435s, table=0, n_packets=171,
n_bytes=16758, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06,nw
_src=10.0.0.1,nw_dst=10.0.0.6
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:1
 cookie=0x10000000c, duration=171.43s, table=0, n_packets=171,
n_bytes=16758, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=1,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01,nw
_src=1.0.0.2,nw_dst=1.0.0.1
actions=mod_nw_src:10.0.0.6,mod_nw_dst:10.0.0.1,output:3
*** s4 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s5 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s6 ----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
```

Desde la interfaz web del controlador, se sigue visualizando la topología virtual de la red (Figura Anexo 70).

Figura Anexo 70 - Alta disponibilidad/Resiliencia – Red virtual desde la interfaz web del controlador

Ahora bien, si se evalúa la interfaz eth-3 del switch 1, se podrá apreciar el tráfico ICMP, lo cual confirma que el mismo se ha derivado por el camino de *backup* (Inicialmente y antes del corte del camino inicial, por este switch no circulaba tráfico).



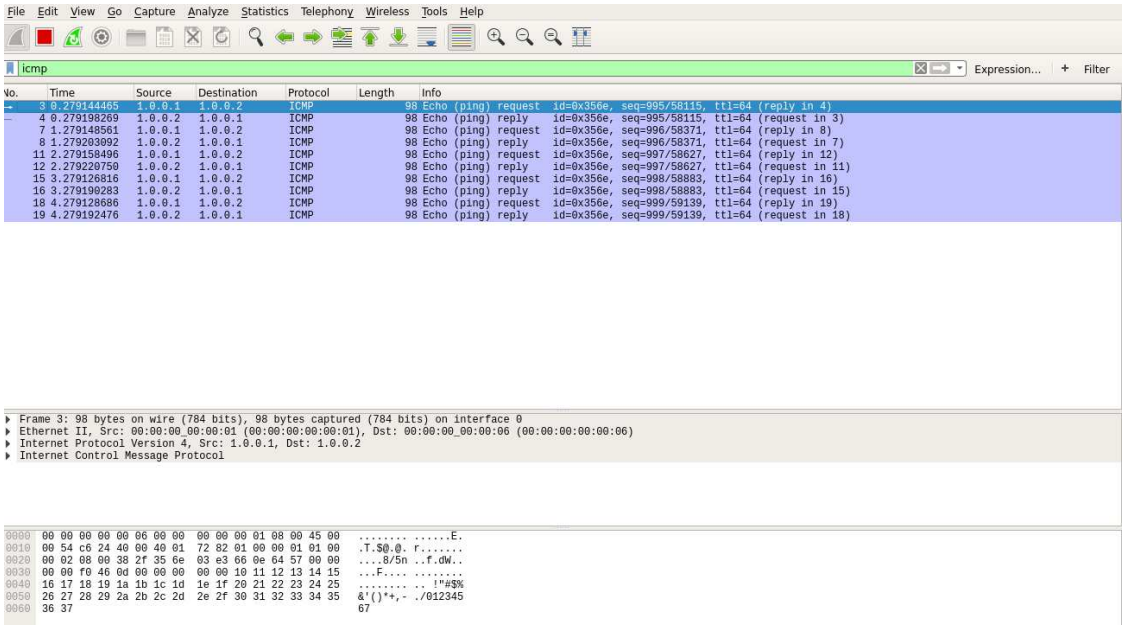


Figura Anexo 71 - Alta disponibilidad/Resiliencia – Tráfico en el puerto 3 del switch 1.

14. ANEXO 14: SIMULACIÓN DE ATRIBUTO AUTOMATIZACIÓN EN EL DESPLIEGUE

OVX tiene una aplicación denominada embedder cuya funcionalidad es la de automatizar el despliegue de redes virtuales.

Se debe acceder al servidor de OVX y dirigirse al path donde se encuentra el ejecutable de la aplicación “embedder”. Por defecto es el siguiente:

~/OpenVirtex/utils/embedder.py

La misma se ejecuta a través del siguiente comando:

python embedder.py

Figura Anexo 72 – Automatización en el despliegue – Ejecución de aplicación embedder

Embedder requiere la descripción de la red virtual a despleguar a través de JavaScript.

A continuación, se muestra un ejemplo utilizado para el despliegue de una red diseñada para esta simulación. En la misma se utiliza un “*bigswitch*”.

Para ello, se debe crear un archivo *.json como el de la Figura Anexo 73.

Figura Anexo 73 – Automatización en el despliegue – JavaScript de red virtual a desplegar en forma automatizada

En la figura anterior se visualiza la configuración de la red virtual a instanciar, la cual tiene las siguientes características:

tenant id=1

controlador=tcp:172.17.0.3:10000

bigswitch= compuesto por los switches físicos cuyo DPID es 00:00:00:00:00:00:00:03, 00:00:00:00:00:00:00:01, 00:00:00:00:00:00:00:02

host= se conectan al bigswitch los host/servidores cuya MAC es 00:00:00:00:00:01, 00:00:00:00:00:05, 00:00:00:00:00:06

Dicho script (Figura Anexo 73) representa a la red “celeste” (Figura Anexo 74).

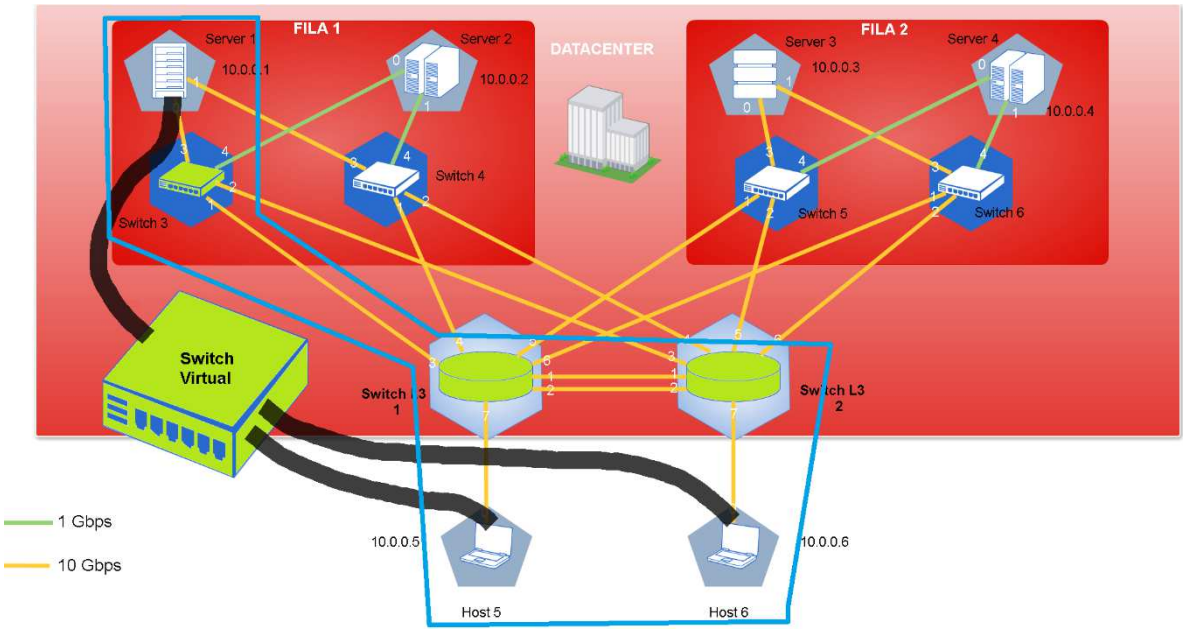


Figura Anexo 74 – Automatización en el despliegue – Red celeste

Luego y con la aplicación embedder en ejecución, se transfiere dicho archivo a la aplicación. Para realizar la transferencia se utiliza *curl* (Figura Anexo 75):

```
root@07d7ddcbf28e:/home# curl localhost:8000 -X POST -d @bigswitch.json
{"jsonrpc": "2.0", "result": {"tenantId": 1}, "id": "1"}
```

Figura Anexo 75 – Automatización en el despliegue – Transferencia del script a través de curl a la aplicación embedder

Una vez realizado esto, el servidor de OVX habrá detectado la configuración correspondiente:

```
2016-06-18 20:00:50,610 _exec_createNetwork
2016-06-18 20:00:50,632 Physical network topology received
2016-06-18 20:00:50,638 Network with tenantId 1 has been created
2016-06-18 20:00:50,644 Switch with switchId 00:a4:23:05:00:00:00:01 has been created
2016-06-18 20:00:50,648 Internal routing of switch 00:a4:23:05:00:00:00:01 has been set to spf
2016-06-18 20:00:50,653 Port on switch 00:a4:23:05:00:00:00:01 with port number 1 has been created
2016-06-18 20:00:50,662 Host with hostId 1 connected
2016-06-18 20:00:50,668 Port on switch 00:a4:23:05:00:00:00:01 with port number 2 has been created
2016-06-18 20:00:50,674 Host with hostId 2 connected
2016-06-18 20:00:50,681 Port on switch 00:a4:23:05:00:00:00:01 with port number 3 has been created
2016-06-18 20:00:50,688 Host with hostId 3 connected
2016-06-18 20:00:50,710 Network with tenantId 1 has been started
127.0.0.1 - - [18/Jun/2016 20:00:50] "POST / HTTP/1.1" 200 -
```

Figura Anexo 76 – Automatización en el despliegue – Vista de la consola de OVX luego de la ejecución del comando de la figura Figura Anexo 75

Luego de ello, en el controlador se puede visualizar la red virtual desplegada de manera “automatizada” (Figura Anexo 77).

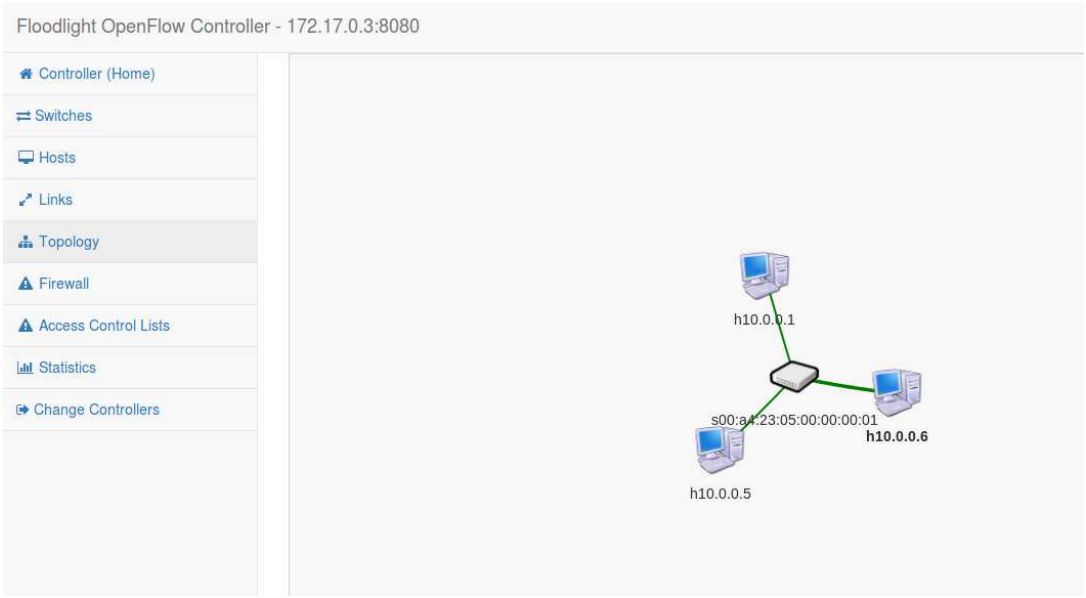


Figura Anexo 77 – Automatización en el despliegue – Red desplegada de manera automatizada

15. ANEXO 15: SIMULACIÓN DE ATRIBUTO GRABAR EL ESTADO DE CONFIGURACIÓN DE UNA RED (SNAPSHOT)

A. OpenVirtex

1) Se emula la red del centro de cómputos (Figura 28) a través de mininet asignando el control de la misma a OVX.

2) Se inicia en el servidor de OVX el servicio de base de datos (MongoDB)

sudo /etc/init.d/mongodb start

Nota: La base de datos MongoDB fue instalada en el servidor de OVX mediante el comando sudo apt-get install mongodb

3) Se inicia OVX. Para ello se ejecuta en ~/OpenVirtex/scripts

#sh ovx.sh

Al iniciar OVX verá el siguiente log, indicando que efectivamente el servidor de OVX está conectado con MongoDB.

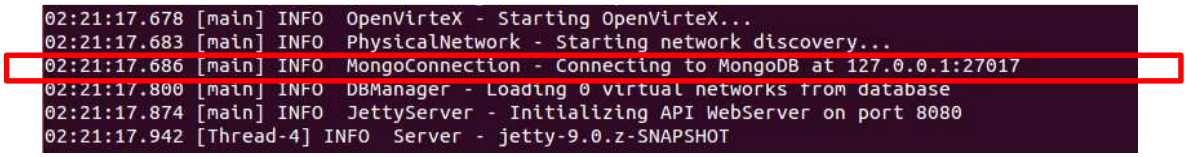


Figura Anexo 78 – Snapshot – Log de inicio de OVX en donde se indica la conexión con la base de datos local

4) Se configura la red naranja utilizada en ANEXO 7: Simulación de atributo de transparencia (Figura Anexo 7).

5) Una vez configurada dicha red, se realiza una prueba de comunicación entre el host 1 y 5 y se visualiza la topología desde el controlador.

6) Se verifica la topología de la red virtual existente a través del siguiente comando:

#python ovxctl.py -n getVirtualTopology 1

{"switches": ["00:a4:23:05:00:00:00:02", "00:a4:23:05:00:00:00:01"], "links": [{"linkId": 1.0, "dst": {"port": "2", "dpid": "00:a4:23:05:00:00:00:01"}, "src": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}, "tenantId": 1.0}, {"linkId": 1.0, "dst": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}, "src": {"port": "2", "dpid": "00:a4:23:05:00:00:00:01"}, "tenantId": 1.0}]}

7) Se procede a interrumpir el servicio de OVX, tal como se muestra en la imagen a continuación (Figura Anexo 79).

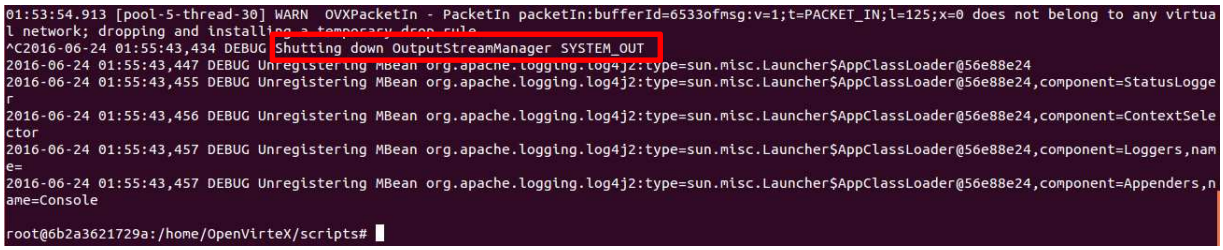


Figura Anexo 79 – Snapshot – Log de la consola de OVX al interrumpir el servicio

8) Se procede a iniciar nuevamente el servicio de OVX. Se puede apreciar en el log que las configuraciones de red virtual son nuevamente restauradas:

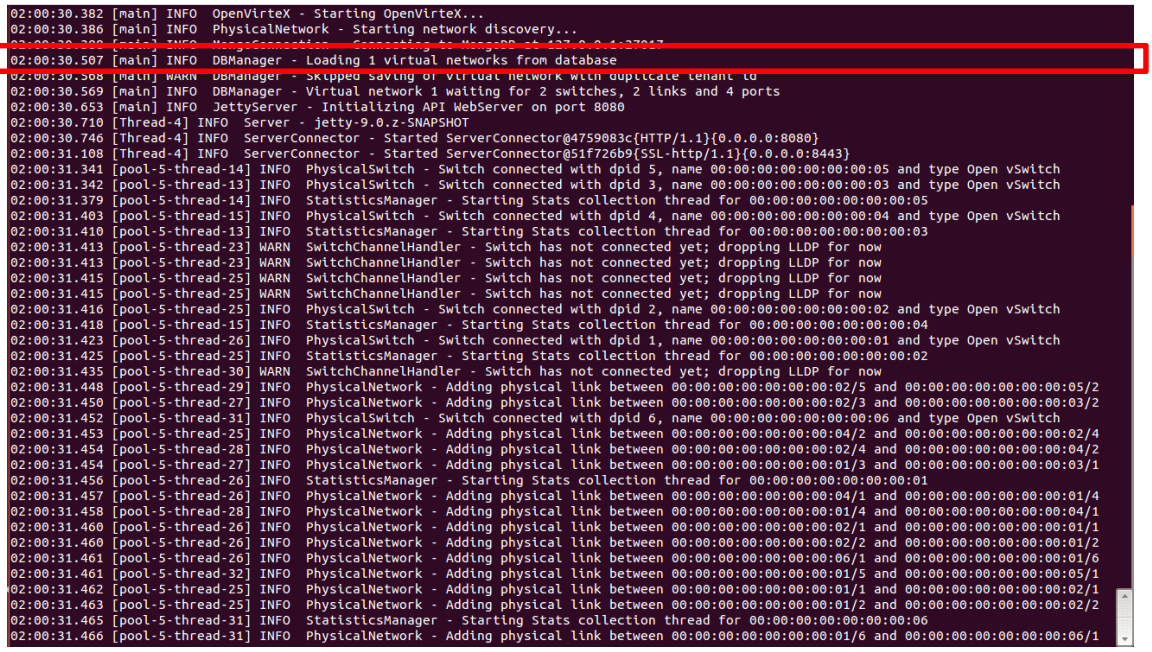


Figura Anexo 80 – Snapshot – Log de inicio de OVX. Se aprecia el proceso de restauración de la red.

9) Se visualiza la configuración de red en el controlador. Se puede apreciar que en forma posterior al reinicio de OVX la topología de red virtual se reconstruye en forma automática a partir de la base de datos.

   Dicho proceso no requiere ningún parámetro ni configuración adicional al momento del reinicio del servicio, puesto que se carga la última configuración existente.

   La topología de red que se visualiza a través del controlador es la presentada en la Figura Anexo 81.

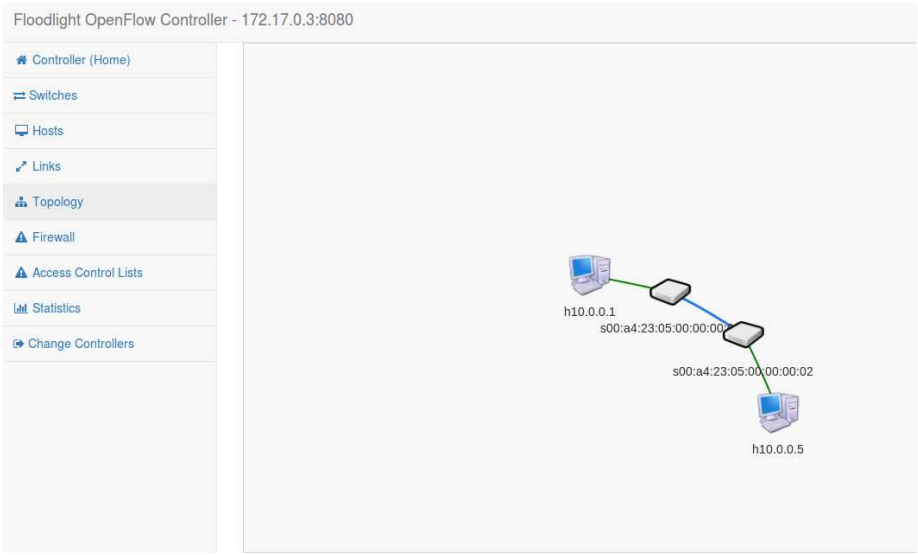


Figura Anexo 81 – Snapshot – Topología de red naranja en forma posterior al reinicio de OVX.

10) Se ejecuta nuevamente el comando para obtener la configuración de la topología de red obteniendo el mismo resultado que el presentado en 6).

#python ovxctl.py -n getVirtualTopology 1

{"switches": ["00:a4:23:05:00:00:00:02", "00:a4:23:05:00:00:00:01"], "links": [{"linkId": 1.0, "dst": {"port": "2", "dpid": "00:a4:23:05:00:00:00:01"}, "src": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}, "tenantId": 1.0}, {"linkId": 1.0, "dst": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}, "src": {"port": "2", "dpid": "00:a4:23:05:00:00:00:01"}, "tenantId": 1.0}]}

11) Si se quisiera eliminar la configuración almacenada en la base datos para que la última configuración no sea restaurada, se debe iniciar nuevamente OVX con el siguiente parámetro.

sh ovx.sh --db-clear

Luego, se puede verificar que efectivamente no existe ninguna configuración en la red de la siguiente manera:

python ovxctl.py -n getVirtualTopology 1

{u'jsonrpc': u'2.0', u'id': u'ovxctl', u'error': {u'message': u'GetVirtualTopology: Invalid tenantId : 1', u'code': -32602}}

Tal como se puede apreciar la configuración preexistente en la base datos no existe más y por lo tanto las redes virtuales se deben reconfigurar nuevamente.

B. FlowVisor

1) Se emula la red del centro de cómputos (Figura 28) a través de mininet, asignando el control de la misma a FV.

2) Se inicia FV y se configura a través de su interfaz de administración la red naranja presentada en ANEXO 7: Simulación de atributo de transparencia.

3) Se realiza una prueba de comunicación entre los hosts 1 y 5 a través de la consola de mininet, para verificar así, el correcto funcionamiento de la red virtual.

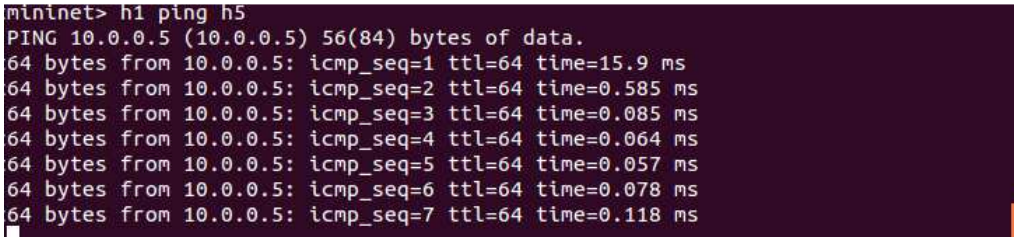


Figura Anexo 82 – Snapshot – ping entre host 1 y 5 correspondientes a la red naranja

4) Se ejecutan los siguientes comandos para visualizar la configuración de *slices* y *flowspaces* existentes en FV (correspondientes a la red “naranja”)

fvctl -f /dev/null list-flowspace

Configured Flow entries:

{"force-enqueue": -1, "name": "dpid3-port3", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:03", "id": 1, "match": {"wildcards": 4194302, "in_port": 3}}

{"force-enqueue": -1, "name": "dpid3-port1", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:03", "id": 2, "match": {"wildcards": 4194302, "in_port": 1}}

{"force-enqueue": -1, "name": "dpid1-port3", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:01", "id": 3, "match": {"wildcards": 4194302, "in_port": 3}}

{"force-enqueue": -1, "name": "dpid1-port7", "slice-action": [{"slice-name": "naranja", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:01", "id": 4, "match": {"wildcards": 4194302, "in_port": 7}}

fvctl -f /dev/null list-slices

Configured slices:

fvadmin --> enabled

naranja --> enabled

5) Luego se procede a grabar la configuración utilizando el comando save-config disponible en la API de gestión.

fvctl save-config config.json

Password:

Config file written to config.json.

Siendo config.json el nombre del archivo *java script* en donde ser guarda la configuración de las redes virtuales instanciadas a través de FV.

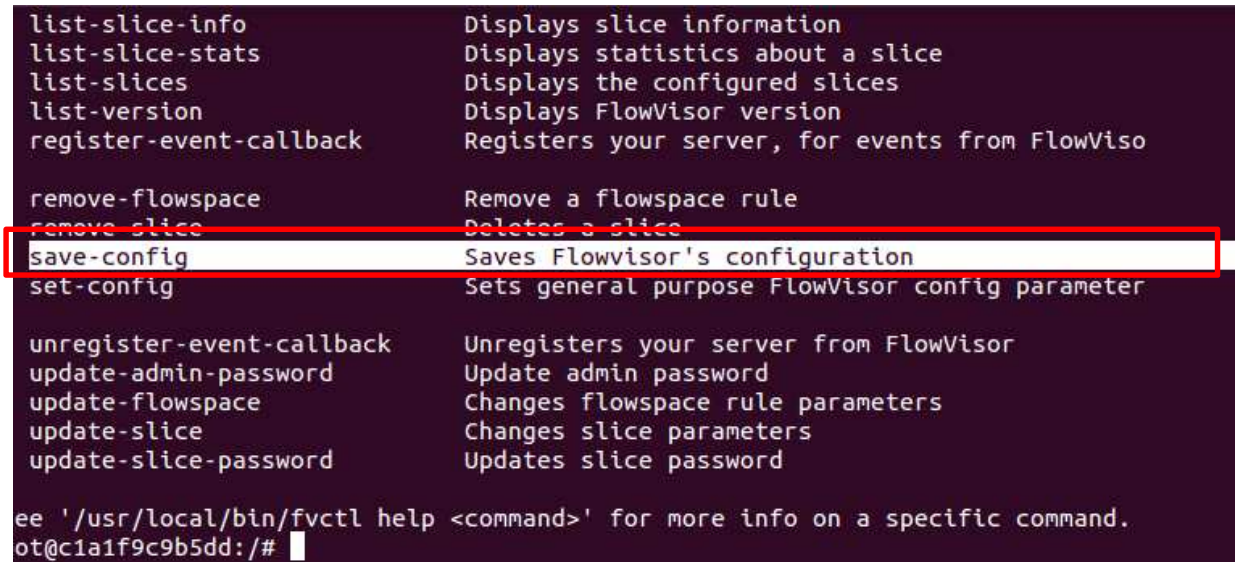


Figura Anexo 83 – Snapshot – Comando save-config disponible a través de la API de administración y configuración de FV (fvctl)

6) Se procede a interrumpir el servidor de FV tal como se puede apreciar en la imagen a continuación:

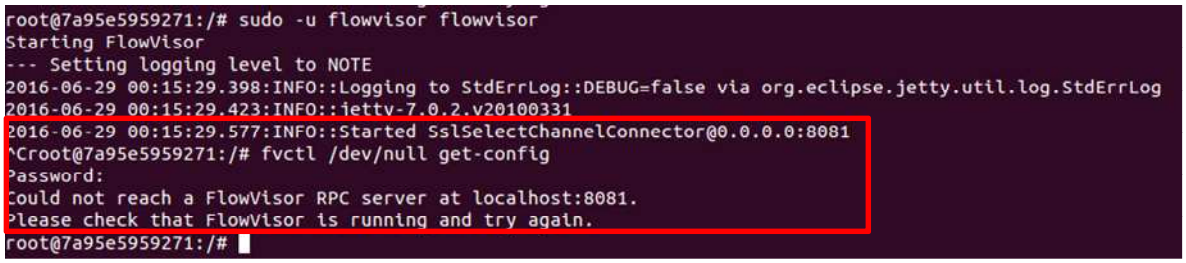


Figura Anexo 84 – Snapshot – Interrupción de servicio de FlowVisor

Luego de ello, se verifica que se pierde la conectividad entre los servidores, como así también la visibilidad de la red desde la interfaz del controlador (Figura Anexo 85).

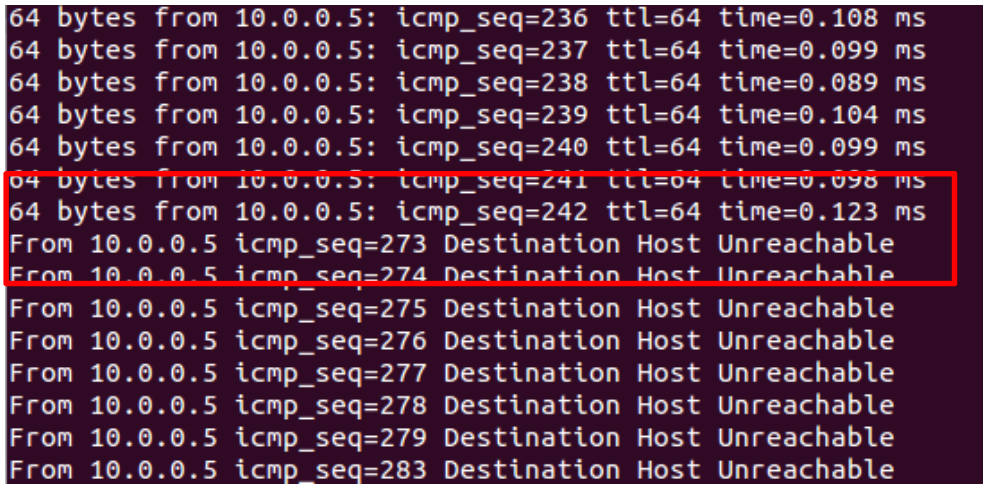


Figura Anexo 85 – Snapshot – Interrupción de servicio de FlowVisor

7) Se procede a iniciar nuevamente el servicio de FlowVisor

En el instante en que se inicia FV, se recupera nuevamente la conectividad entre los dos hosts de la red naranja (host 1 y host 5) de manera automática. No se requiere la ejecución de ningún comando ni configuración adicional y por defecto se vuelve a configurar red “naranja”.

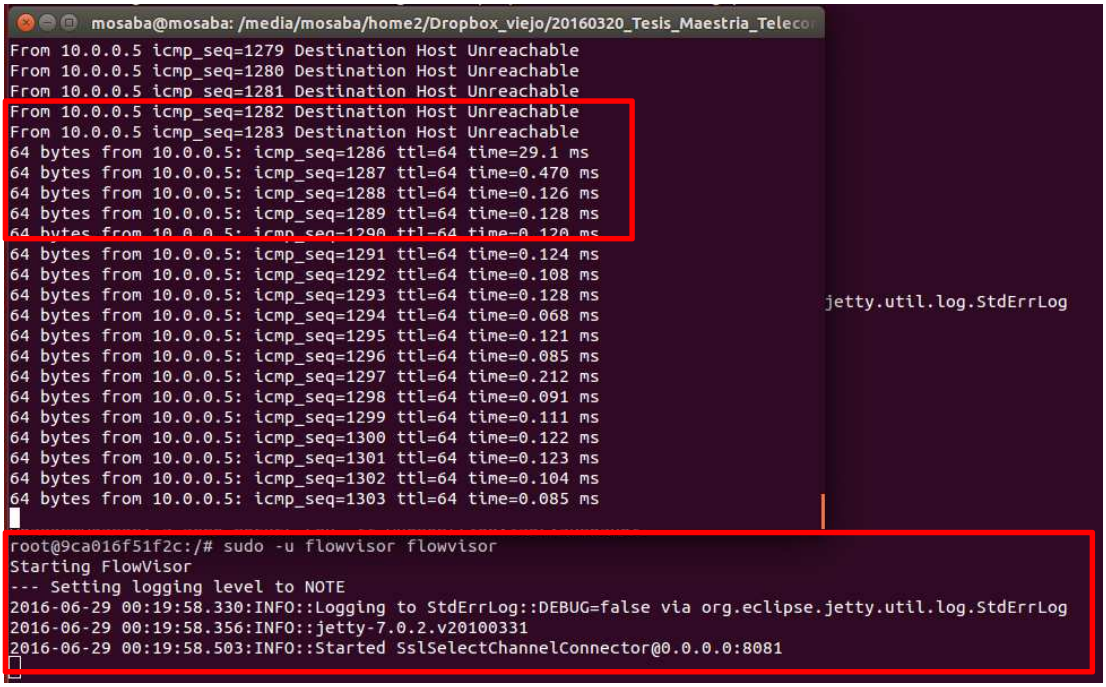


Figura Anexo 86 – Snapshot – Inicio del servicio de FlowVisor

8) Se verifica que la configuración de la red sea la correcta. Para ello se ejecuta el comando fvctl list-flowspace y se compara con el presentado en el punto 4).

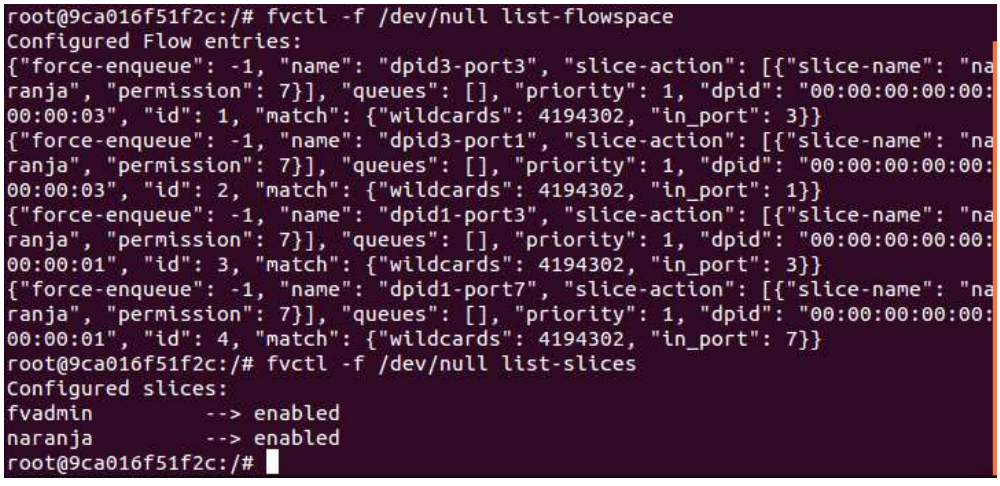


Figura Anexo 87 – Snapshot – Configuración luego del inicio del servicio de FV

Dicha configuración se visualiza a través de la interfaz web del controlador.

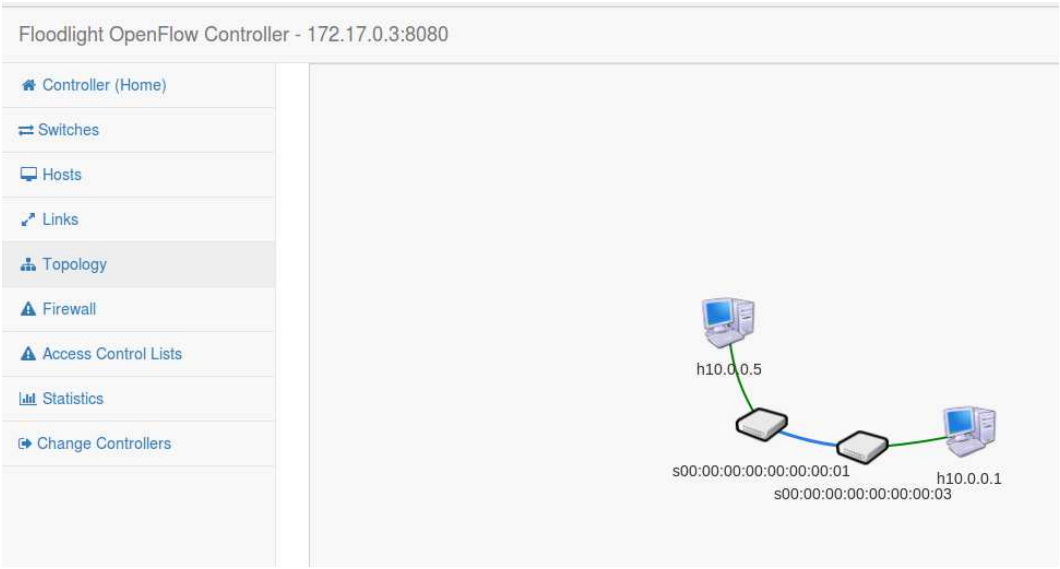


Figura Anexo 88 - Snapshot – Configuración luego del inicio del servicio de FV

16.ANEXO 16: SIMULACIÓN DE ATRIBUTO VIRTUALIZACIÓN RECURSIVA

Se diseñan las redes “gris” y “naranja”. La configuración de la red “naranja” se expone en ANEXO 7: Simulación de atributo de transparencia.

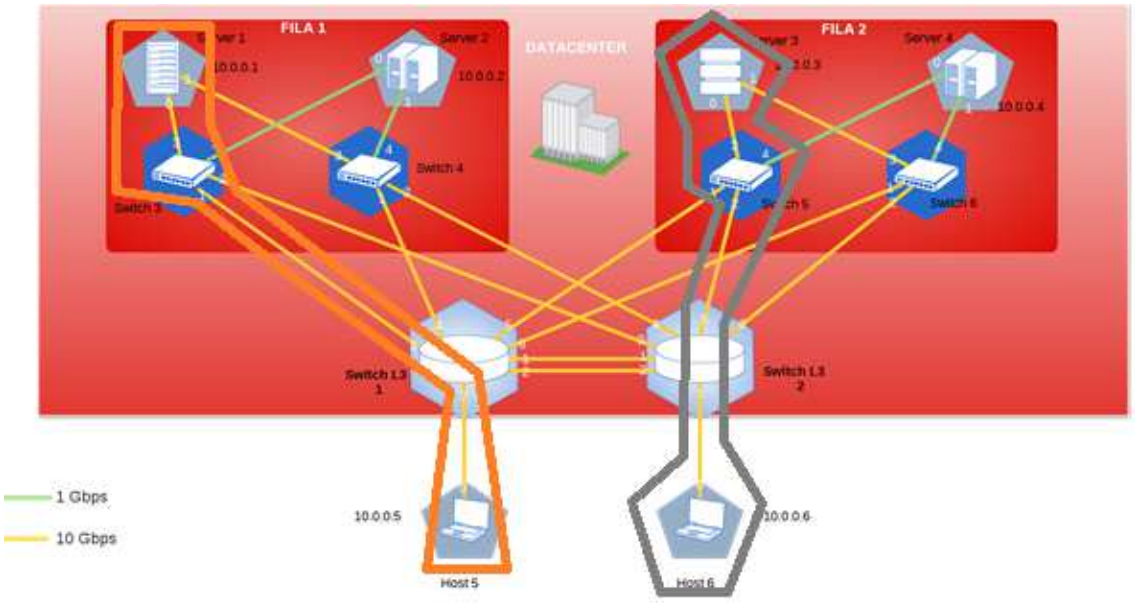


Figura Anexo 89 – Virtualización recursiva – Diseño de red gris y naranja.

A. OpenVirtex

Se ejecutan dos instancias de OVX y dos instancias del controlador FloodLight. El diseño lógico de la simulación se presenta en la Figura Anexo 90.

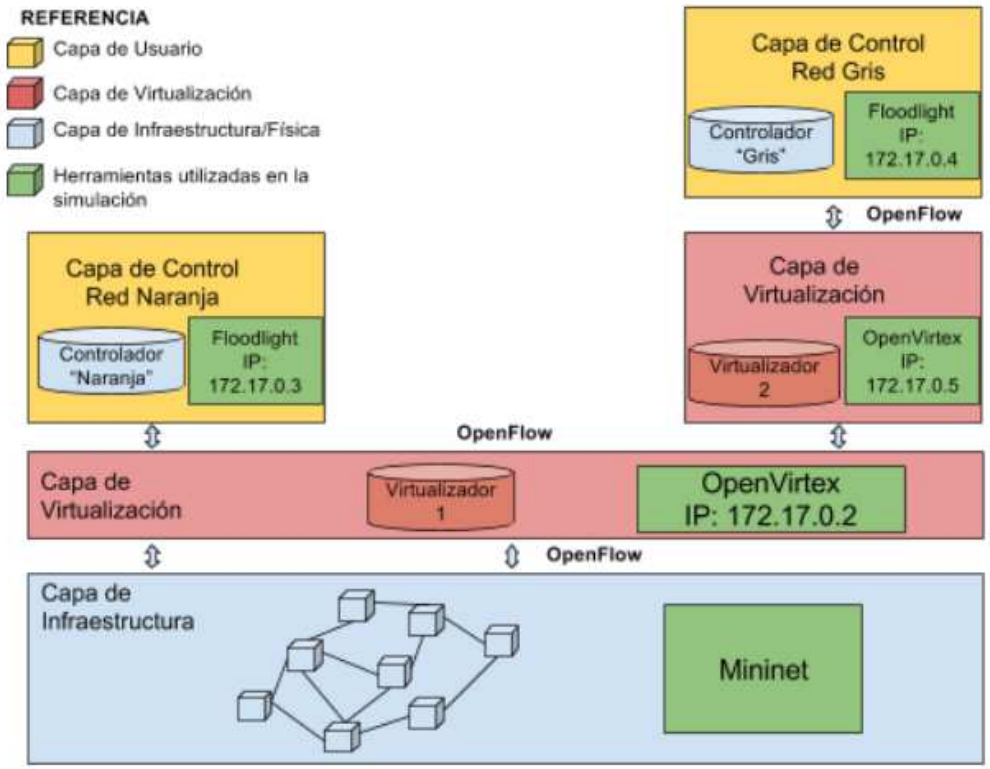


Figura Anexo 90 – Virtualización Recursiva – Diseño lógico de la simulación

Si se ingresa en la terminal de la PC de pruebas el siguiente comando:

sudo docker ps

Se obtiene la siguiente respuesta:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

aa3fe8cddb10 mno808/openvirtex:v3 "bash" 6 minutes ago Up 6 minutes amazing_dijkstra

873e2dc17560 mno808/floodlight:v0 "bash" 2 hours ago Up 2 hours amazing_leavitt

04bb3070d57e mno808/floodlight:v0 "bash" 2 hours ago Up 2 hours happy_yonath

5f7aec13fa0b mno808/openvirtex:v3 "bash" 2 hours ago Up 2 hours thirsty_goldstine

Se puede apreciar que se encuentran ejecutándose cuatro instancias de dockers:

La primera instancia de OVX con container ID: 5f7aec13fa0b es la correspondiente a la IP 172.17.0.2 y se utiliza para virtualizar las redes “gris” y “naranja”.

La red naranja se asigna al controlador con container ID: 04bb3070d57e, IP 172.17.0.3 (el detalle de la configuración se encuentra en ANEXO 7: Simulación de atributo de transparencia).

La red gris se asigna a la segunda instancia de OVX con container ID: aa3fe8cddb10 y dirección IP 172.17.0.5. Dicha red gris se virtualiza en forma recursiva y se asigna al controlador con container ID: aa3fe8cddb10, IP 172.17.0.4.

A continuación, se exponen las configuraciones correspondientes:

Configuración de la red “gris”

1) Creación de la red:

python ovxctl.py -n createNetwork tcp:172.17.0.5:6633 10.0.0.0 16

Virtual network has been created (network_id {u'mask': 16, u'networkAddress': 167772160, u'controllerUrls': [u'tcp:172.17.0.5:6633'], u'tenantId': 2}).

Recordar que la la dirección de IP 172.17.0.5 corresponde a una segunda instancia de OVX.

2) Creación de los switches virtuales:

python ovxctl.py -n createSwitch 2 00:00:00:00:00:00:00:05

Virtual switch has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01)

python ovxctl.py -n createSwitch 2 00:00:00:00:00:00:00:02

Virtual switch has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02)

3) Creación de los puertos virtuales:

```
# python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:05 2
Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01, port_id 1)
# python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:05 3
Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:01, port_id 2)
# python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:02 5
Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02, port_id 1)
# python ovxctl.py -n createPort 2 00:00:00:00:00:00:00:02 7
Virtual port has been created (tenant_id 2, switch_id 00:a4:23:05:00:00:00:02, port_id 2)
```

4) Creación de los links virtuales:

```
# python ovxctl.py -n connectLink 2 00:a4:23:05:00:00:00:01 1 00:a4:23:05:00:00:00:02
1 sfp 1
Virtual link (link_id 1) has been created
```

5) Conexión de los hosts:

```
# python ovxctl.py -n connectHost 2 00:a4:23:05:00:00:00:01 2 00:00:00:00:00:03
Host (host_id 1) has been connected to virtual port
# python ovxctl.py -n connectHost 2 00:a4:23:05:00:00:00:02 2 00:00:00:00:00:06
Host (host_id 2) has been connected to virtual port
```

6) Inicio de la red:

```
# python ovxctl.py -n startNetwork 2
Network (tenant_id 2) has been booted
```

Desde la consola de la segunda instancia de OVX (IP 172.17.0.5) se ejecutan los siguientes comandos:

```
# python ovxctl.py -n getPhysicalTopology
```

{"switches": ["00:a4:23:05:00:00:00:02", "00:a4:23:05:00:00:00:01"], "links": [{"linkId": 0.0, "dst": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}, "src": {"port": "1", "dpid": "00:a4:23:05:00:00:00:01"}}, {"linkId": 1.0, "dst": {"port": "1", "dpid": "00:a4:23:05:00:00:00:01"}, "src": {"port": "1", "dpid": "00:a4:23:05:00:00:00:02"}}]}

```
# python ovxctl.py -n getVirtualHosts 1
```

[{"mac": "00:00:00:00:00:03", "hostId": 1.0, "ipAddress": "10.0.0.1", "port": 1.0, "dpid": "00:a4:23:05:00:00:00:01"}, {"mac": "00:00:00:00:00:06", "hostId": 2.0, "ipAddress": "10.0.0.5", "port": 2.0, "dpid": "00:a4:23:05:00:00:00:02"}]

El *output* de los comandos ejecutados, indica que la segunda instancia de OVX visualiza a la red virtual “gris” de acuerdo a la Figura Anexo 91.

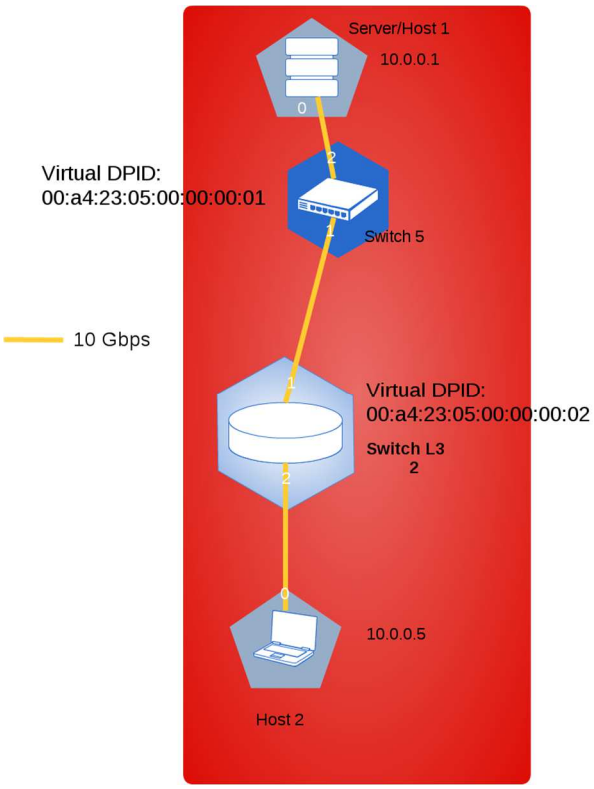


Figura Anexo 91 – Virtualización Recursiva – Red “gris” vista desde la segunda instancia de OVX (IP 172.17.0.5)

Desde la consola de la segunda instancia de OVX (IP 172.17.0.5), se virtualiza recursivamente la red virtual gris y se procede a asignar la misma al controlador Floodlight IP 172.17.0.4 (Figura Anexo 90).

Configuración de la red "gris" virtualizada recursivamente:

1) Creación de la red virtual

python ovxctl.py -n createNetwork tcp:172.17.0.4:20000 10.0.0.0 16

Virtual network has been created (network_id {u'mask': 16, u'networkAddress': 167772160, u'controllerUrls': [u'tcp:172.17.0.4:20000'], u'tenantId': 1}).

2) Creación de los switches virtuales

python ovxctl.py -n createSwitch 1 00:a4:23:05:00:00:00:01

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

python ovxctl.py -n createSwitch 1 00:a4:23:05:00:00:00:02

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02)

3) Creación de los puertos virtuales

python ovxctl.py -n createPort 1 00:a4:23:05:00:00:00:01 2

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

python ovxctl.py -n createPort 1 00:a4:23:05:00:00:00:01 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

python ovxctl.py -n createPort 1 00:a4:23:05:00:00:00:02 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 1)

python ovxctl.py -n createPort 1 00:a4:23:05:00:00:00:02 2

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:02, port_id 2)

4) Conexión de los switches virtuales

```
# python ovxctl.py -n connectLink 1 00:a4:23:05:00:00:00:01 2 00:a4:23:05:00:00:00:02
1 sfp 1
Virtual link (link_id 1) has been created
```

5) Conexión de los host

```
python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:03
Host (host_id 1) has been connected to virtual port
python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:02 2 00:00:00:00:00:06
Host (host_id 2) has been connected to virtual port
```

6) Inicio de la red

```
# python ovxctl.py -n startNetwork 1
Network (tenant_id 1) has been booted
```

Desde el controlador Floodlight IP 172.17.0.4 se puede apreciar la red gris virtualizada recursivamente tal como se aprecia en la Figura Anexo 92.

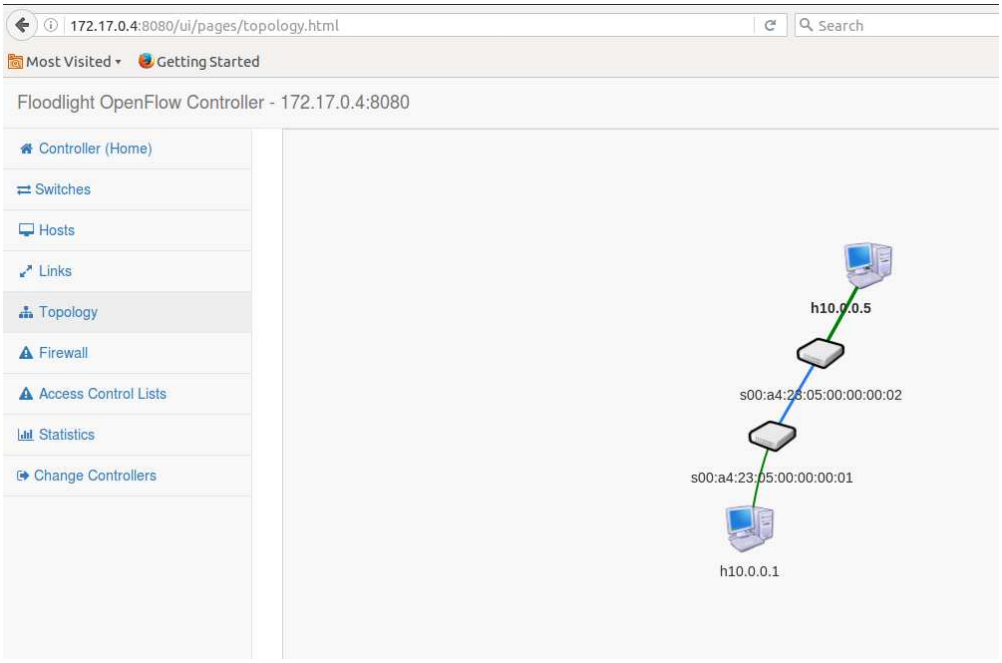


Figura Anexo 92 – Virtualización Recursiva – Red gris virtualizada recursivamente

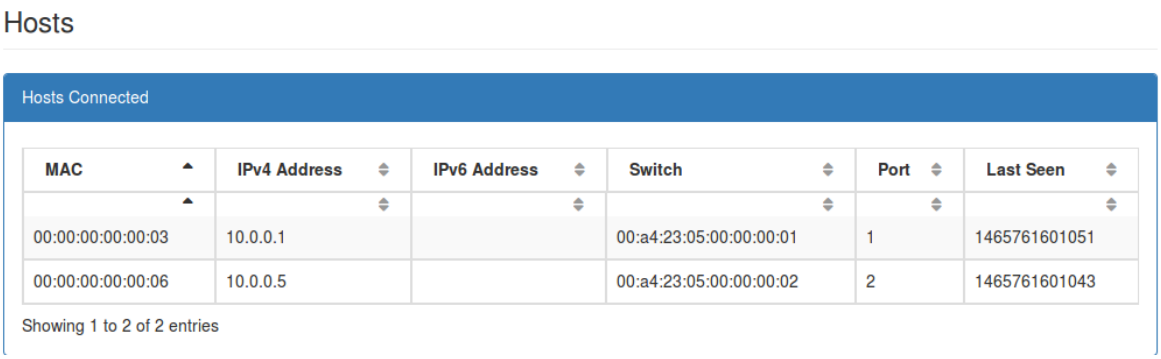


Figura Anexo 93 – Virtualización Recursiva – Hosts asociados a la red gris virtualizada recursivamente

B. FlowVisor

En el caso de FV no se instancia la red virtual naranja presentada en la Figura Anexo 89. Se identifica al momento de realizar la experiencia mediante OVX que este paso no resulta necesario para verificar el atributo de virtualización recursiva.

La configuración lógica utilizada es la que se muestra en la imagen a continuación:

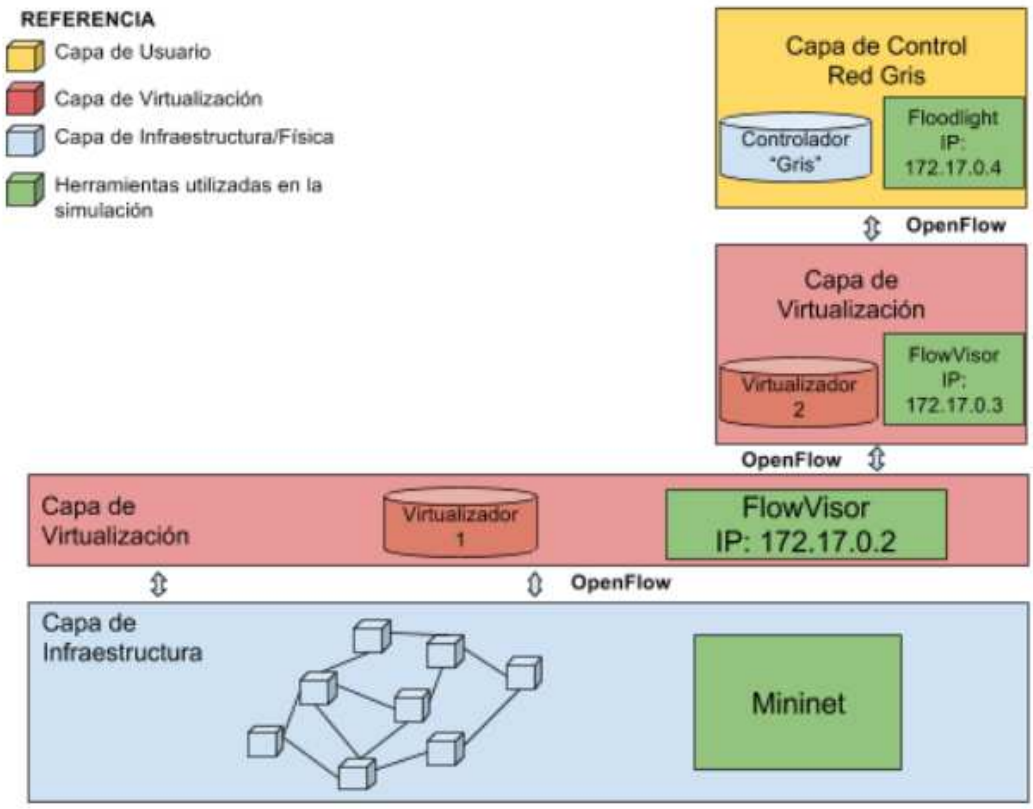


Figura Anexo 94 – Virtualización Recursiva – Diseño lógico de la simulación con FV

Si se ingresa en la terminal de la PC de pruebas el siguiente comando:

sudo docker ps

Se obtiene la siguiente respuesta:

```
mosaba@mosaba:~$ sudo docker ps
[sudo] password for mosaba:
CONTAINER ID        IMAGE                       COMMAND             CREATED
    STATUS              PORTS                   NAMES
8a5f9928fe07        mno808/floodlight:v0        "bash"              22 minutes ago
    Up 22 minutes                               backstabbing_heisenberg
dbd8d62b78b0        mno808/flowvisor:v2.0       "/bin/bash"         23 minutes ago
    Up 23 minutes                               grave_jang
df9c0ac7fbf0        mno808/flowvisor:v2.0       "/bin/bash"         24 minutes ago
    Up 24 minutes                               cocky_austin
mosaba@mosaba:~$
```

Figura Anexo 95 - Virtualización Recursiva – Dockers en ejecución FV

Se puede apreciar que están ejecutándose tres instancias de dockers:

La primera instancia de FV con container ID: df9c0ac7fbf0 es la correspondiente a la IP 172.17.0.2 y se utiliza para virtualizar las redes “gris”.

La red gris se asigna a la segunda instancia de FV con container ID: dbd8d62b78b0 y dirección IP 172.17.0.3. Dicha red gris se virtualiza en forma recursiva y se asigna al controlador con container ID: 8a5f9928fe07, IP 172.17.0.4.

A continuación, se exponen las configuraciones correspondientes:

1) Se inicia la red emulada en mininet y se le asigna a la primera instancia de FV (IP=172.17.0.2).

sudo mn –controller=remote,ip=172.17.0.2 –link tc –custom=labo_topo1.py –topo=redtopo –mac

Configuración de la red “gris”

Desde la consola de FV IP 172.17.0.2 se configura la red gris.

2) Creación de la red virtual gris.

fvctl -f /dev/null add-slice gris tcp:172.17.0.3:6633 admin@gris
Slice password:
Slice gris was successfully created

La IP 172.17.0.3 corresponde a la segunda instancia de FV. El puerto OF por defecto es el 6633.

3) Se configura la red virtual gris.

#fvctl -f /dev/null add-flowspace dpid5-port3 5 1 in_port=3 gris=7
fvctl -f /dev/null add-flowspace dpid5-port2 5 1 in_port=2 gris=7
FlowSpace dpid5-port2 was added with request id 1.
fvctl -f /dev/null add-flowspace dpid2-port5 2 1 in_port=5 gris=7

FlowSpace dpid2-port5 was added with request id 2.

fvctl -f /dev/null add-flowspace dpid2-port7 2 1 in_port=7 gris=7

FlowSpace dpid2-port7 was added with request id 3.

4) Se ejecuta el comando get-config a través de la consola de configuración de la primera instancia de FV (IP 172.17.0.2).

El output de este comando se presenta en la figura a continuación.

Figura Anexo 96 – Virtualización Recursiva – Salida del comando get-config de la primera instancia de FV (IP 172.17.0.2).

Se aprecia que se “visualizan” todos los elementos de red física creada en mininet:

5) Se ejecuta el comando get-config desde la consola de la segunda instancia de FV.

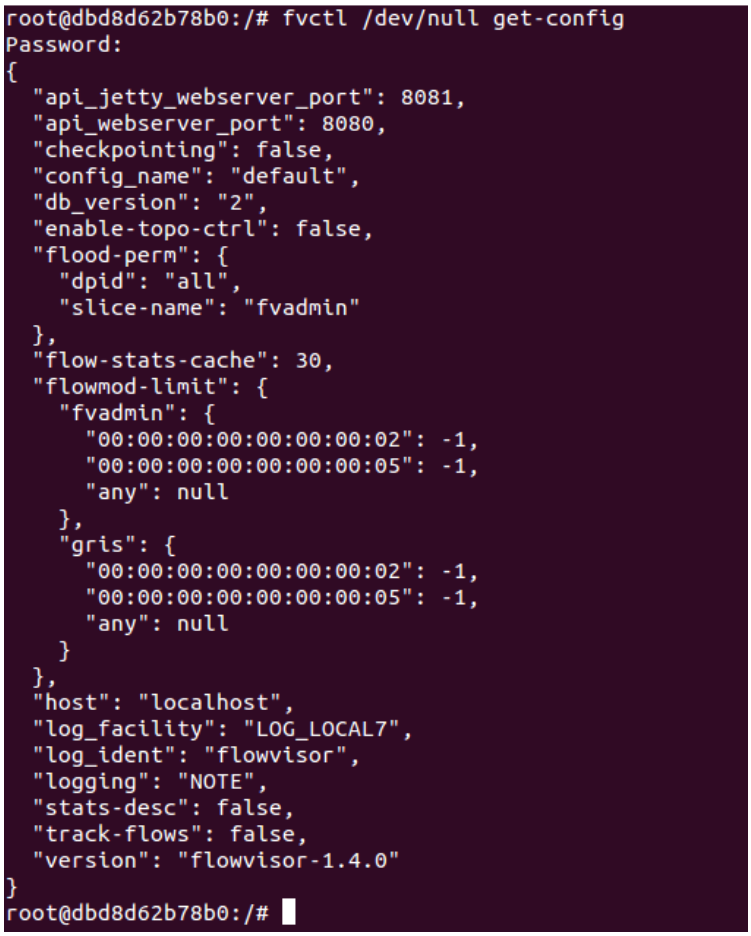


Figura Anexo 97 - Virtualización Recursiva – Salida del comando get-config de la primera segunda instancia de FV (IP 172.17.0.3)

La segunda instancia de FV visualiza solamente los switches correspondientes a la red “gris”.

Configuración de la red “gris” virtualizada recursivamente:

6) Desde la consola de la segunda instancia de FV, se crea la red virtual gris en forma recursiva y se asigna al controlador Floodlight (IP 172.17.0.4).

fvctl -f /dev/null add-slice gris tcp:172.17.0.4:10000 admin@gris

Slice password:

Slice gris was successfully created

7) Se configura la red virtual gris.

```
# fvctl -f /dev/null add-flowspace dpid5-port3 5 1 in_port=3 gris=7
# fvctl -f /dev/null add-flowspace dpid5-port2 5 1 in_port=2 gris=7
FlowSpace dpid5-port2 was added with request id 1.
# fvctl -f /dev/null add-flowspace dpid2-port5 2 1 in_port=5 gris=7
FlowSpace dpid2-port5 was added with request id 2.
# fvctl -f /dev/null add-flowspace dpid2-port7 2 1 in_port=7 gris=7
FlowSpace dpid2-port7 was added with request id 3.
```

8) Se ejecuta un ping entre los hosts 3 y 6. Se puede apreciar que hay comunicación entre ellos.

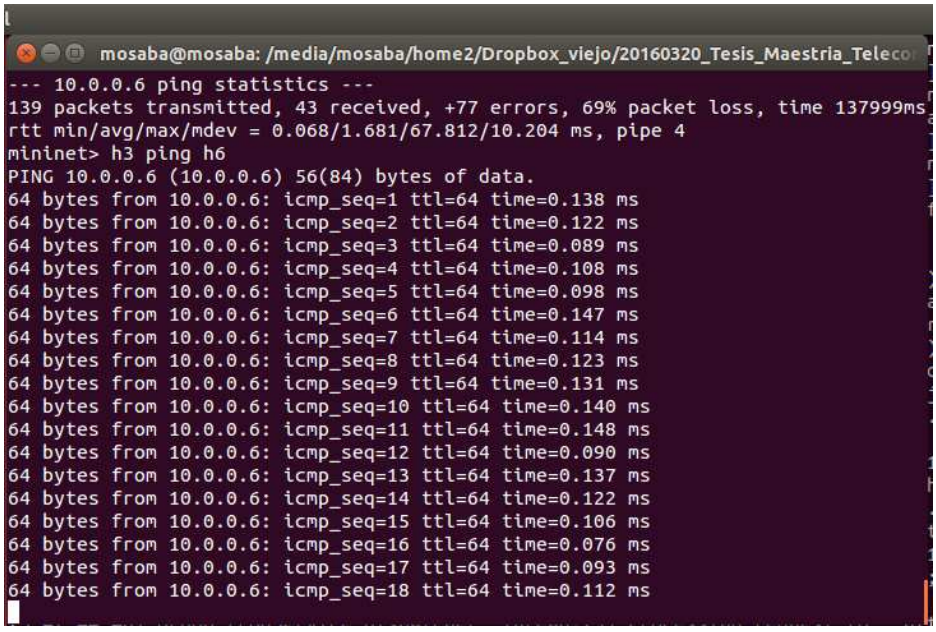


Figura Anexo 98 – Virtualización Recursiva – ping entre hosts 3 y 6 asignados a la red virtual gris recursiva

9) Se accede a la interfaz web del controlador donde se visualiza la red gris recursiva.

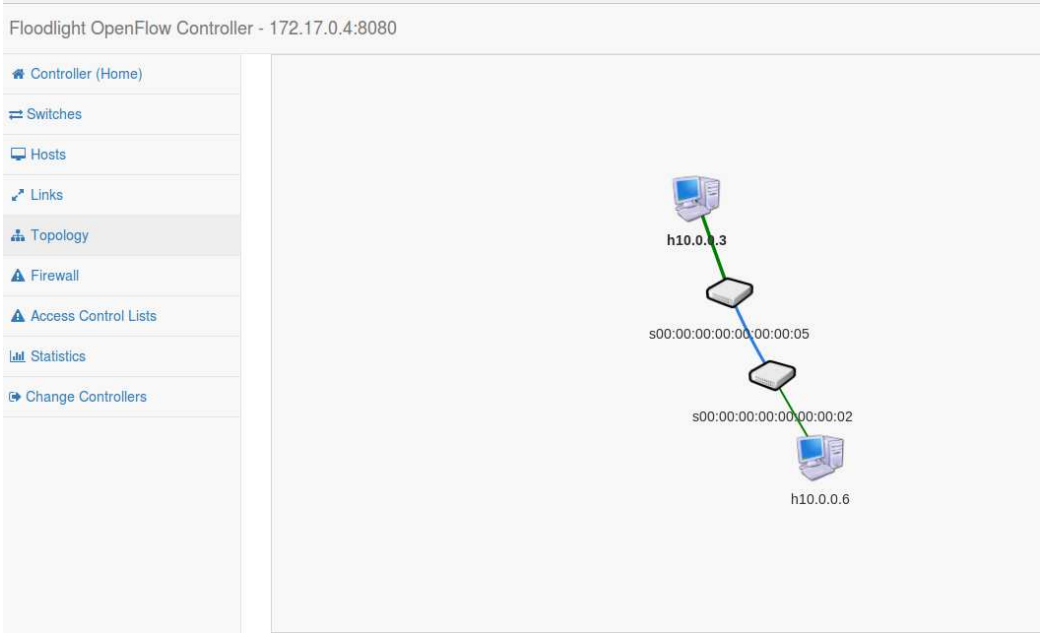


Figura Anexo 99 – Virtualización Recursiva – Red gris virtualizada recursivamente desde el controlador (IP 172.17.0.4)

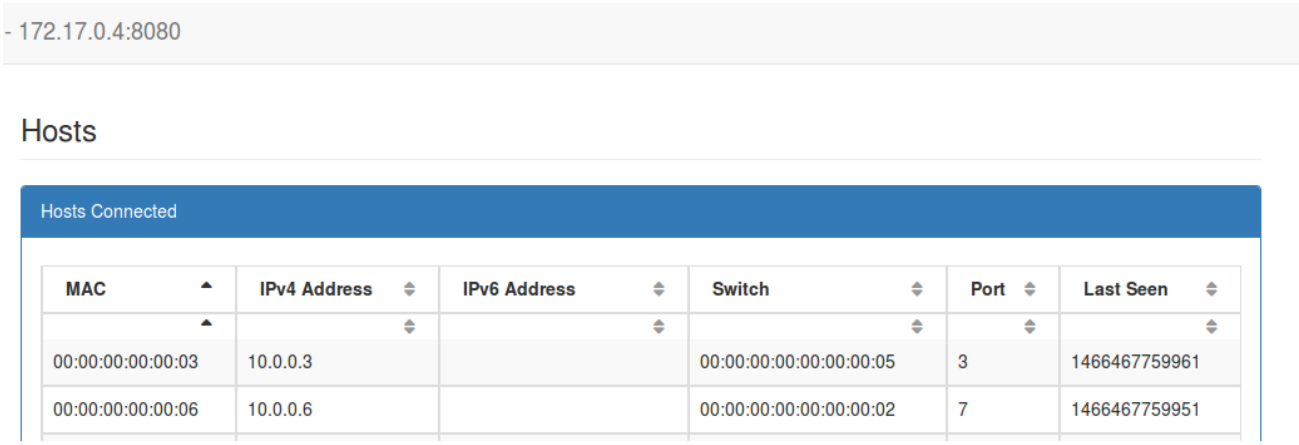


Figura Anexo 100 – Virtualización Recursiva - Hosts asociados a la red gris virtualizada recursivamente FV

17.ANEXO 17: FUNCIONAMIENTO EN REDES HÍBRIDAS

A. Configuración de la red emulada en GNS3

Para configurar la red emulada en GNS3 se requiere del router a utilizar con el OS correspondiente cargado.

Luego, se coloca en el workspace el router y el host, el cual representa a la PC física. Dicho host se configura para que “exponga” su interfaz física eth0, la cual se conecta con una interfaz del router.

Dicha interfaz se configura haciendo doble click sobre el router e insertando los comandos correspondientes a través del CLI.

Configuración del router

```
R1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.

R1(config)#interface  ethernet 1/0
R1(config-if)#

R1(config-if)#no shutdown

R1(config-if)#ip address dhcp

R1(config-if)#exit
R1(config)#exit

R1#wr
```

A continuación, se presenta la red emulada:

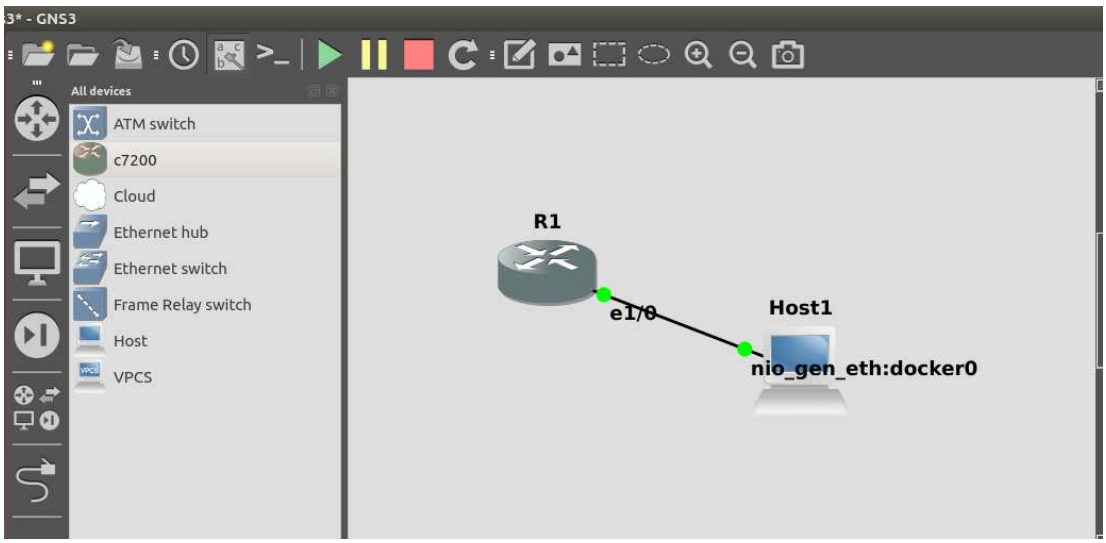


Figura Anexo 101 – Funcionamiento en redes híbridas – Red emulada en GNS3

B. Configuración de la red emulada en mininet

La simulación se realiza en base a una red definida por un switch y un host. A diferencia de las simulaciones realizadas para evaluar otros atributos, en este caso se configura mininet para que el switch OVS tenga conexión con la red física, es decir hacia afuera del dominio de mininet.

Para ello, se toma como referencia un script [74] que configura el switch virtual con una interfaz conectada a la interfaz física eth0 de la PC de prueba. Dicho script, se modifica de acuerdo al diseño de red deseado; se expone a continuación:

```
#!/usr/bin/python


from mininet.net import Mininet

from mininet.node import Controller

from mininet.node import RemoteController

from mininet.cli import CLI

from mininet.link import Intf

from mininet.log import setLogLevel, info


def myNetwork():


   net = Mininet( topo=None,

                  build=False)


   info( '*** Adding controller\n' )
```

```
    c0 = net.addController('c0', controller=RemoteController, ip='172.17.0.2',
port=6633)


    info( '*** Add switches\n')
    s1 = net.addSwitch('s1')
    Intf( 'eth0', node=s1 )


    info( '*** Add hosts\n')
    h1 = net.addHost('h1', ip='0.0.0.0')


    info( '*** Add links\n')
    net.addLink(h1, s1)


    info( '*** Starting network\n')
    net.start()
    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

La línea c0 = net.addController('c0', controller=RemoteController, ip='172.17.0.2', port=6633), indica que el controlador que administrará la red será OVX o FV cuya ip será 172.17.0.2.

Una vez creada la red en mininet, se procede a virtualizar la red.

C. OpenVirtex: Virtualización de la red

1) Creación de la red virtual

python ovxctl.py -n createNetwork tcp:172.17.0.3:10000 192.168.1.0 16

Virtual network has been created (network_id {u'mask': 16, u'networkAddress': -1062731520, u'controllerUrls': [u'tcp:172.17.0.3:10000'], u'tenantId': 1}).

La dirección 172.17.0.3:10000 corresponde a la dirección IP y al puerto donde se encuentra ejecutándose el controlador FloodLight.

2) Creación del switch virtual correspondiente al único switch físico creado a través de la red emulada con Mininet

python ovxctl.py -n createSwitch 1 00:00:00:00:00:00:00:01

Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)

3) Creación de los dos puertos virtuales correspondientes a los puertos físicos utilizados por el switch en el diseño de red

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 1

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 1)

python ovxctl.py -n createPort 1 00:00:00:00:00:00:00:01 2

Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)

En el puerto 1 del switch emulado en Mininet se encuentra conectada la red de datos (router emulado en GNS3, AP con acceso a internet y el End-Point físico conectado a la red WiFi), mientras que en el puerto 2 del mismo, el host emulado por Mininet.

4) Se configura la conexión entre los host y el switch virtual

```
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 1 94:b1:0a:e1:68:4e
Host (host_id 1) has been connected to virtual port
# python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 2 f2:f4:ae:f5:58:1c
Host (host_id 2) has been connected to virtual port
```

En verde se resaltan las direcciones MAC de los dispositivos físicos a conectar al switch virtual. La dirección MAC 94:b1:0a:e1:68:4e corresponde al dispositivo móvil conectado al AP WiFi, tal como se puede apreciar en las imágenes a continuación:

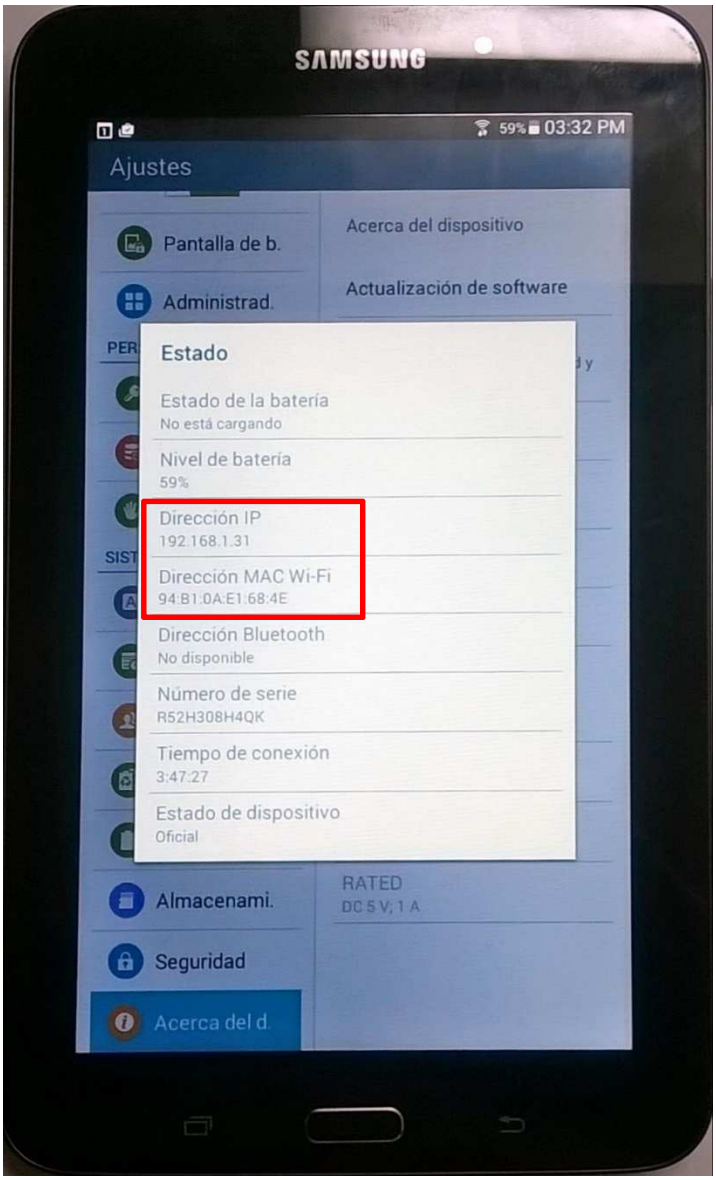


Figura Anexo 102 – Funcionamiento en redes híbridas – Dirección IP y MAC del dispositivo móvil conectado al AP

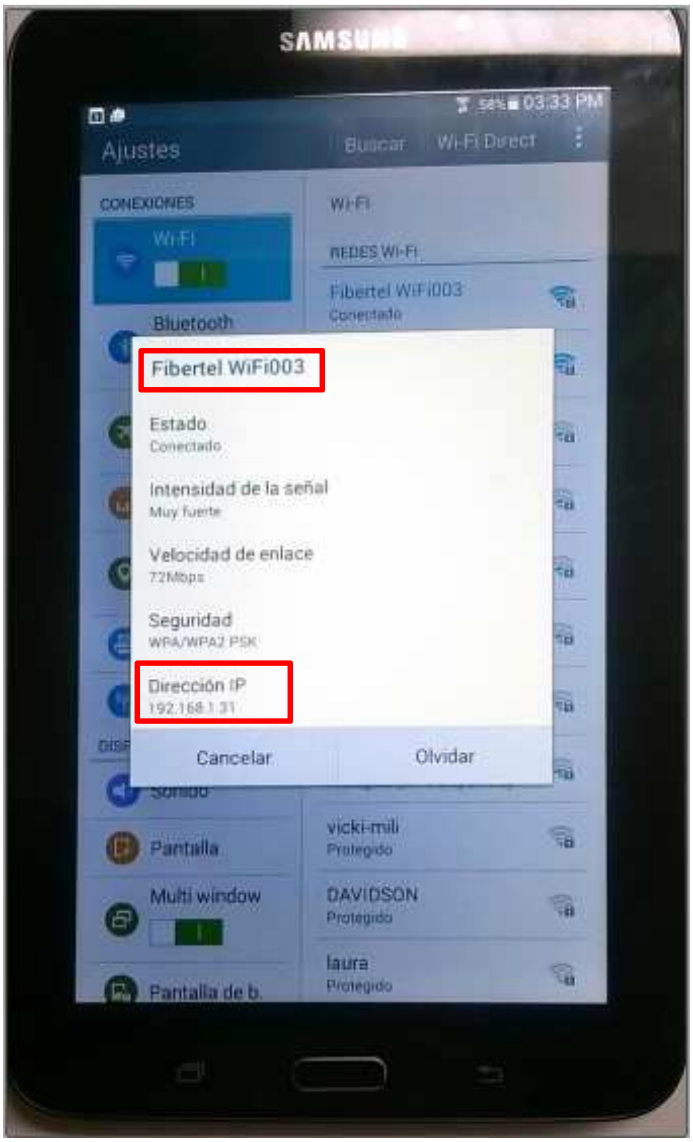


Figura Anexo 103 - Funcionamiento en redes híbridas – SSID de red WiFi y dirección IP del dispositivo móvil conectado al AP

Se debe destacar, que el switch virtual permite la comunicación de este dispositivo a través de su MAC (regla de flujo) abstrayéndose de los dispositivos de red físicos existentes en la red LAN/WAN, que exceden al dominio de OpenFlow (acotado a la red emulada a través de Mininet).

La dirección MAC f2:f4:ae:f5:58:1c corresponde al host virtual creado por Mininet tal como se aprecia a continuación:

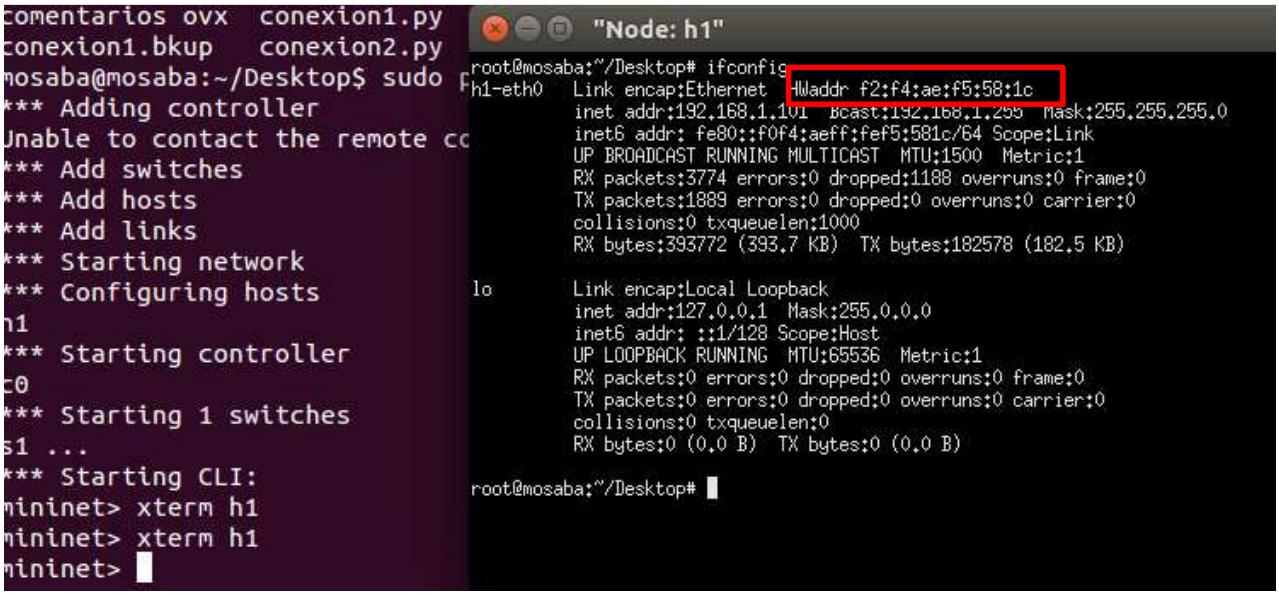


Figura Anexo 104 - Funcionamiento en redes híbridas – Dirección MAC del host emulado a través de mininet

5) Se inicia la red virtual configurada.

python ovxctl.py -n startNetwork 1

Network (tenant_id 1) has been booted

6) Una vez finalizada la configuración se procede a realizar un ping desde el host emulado en mininet (h1) hacia el dispositivo móvil conectado a la red WiFi. Tal como se muestra en la imagen a continuación, la red virtual se configuró correctamente siendo posible la comunicación entre los dos hosts.

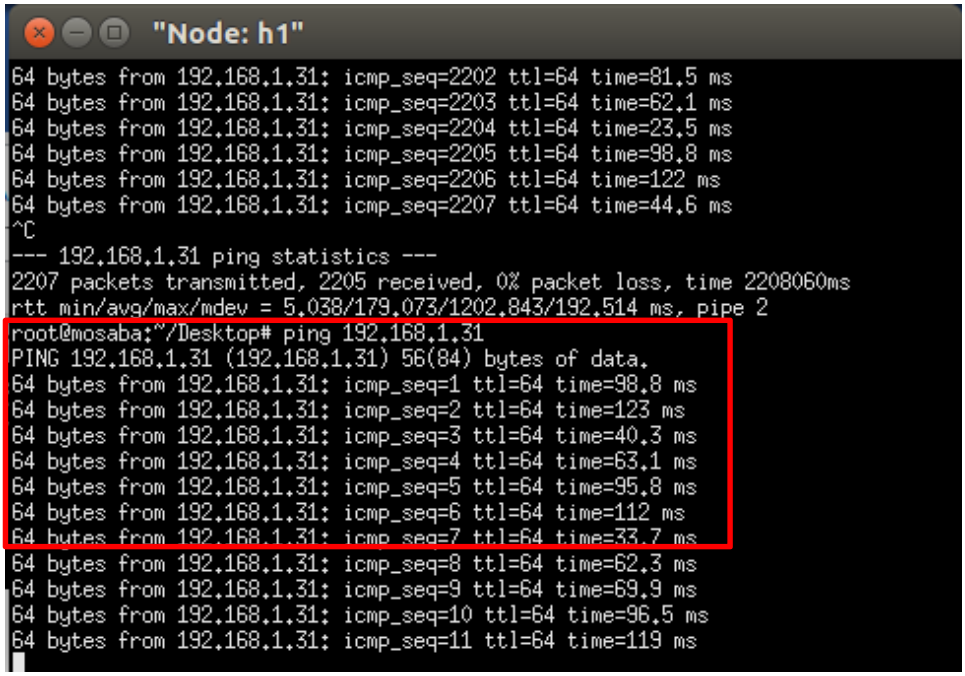


Figura Anexo 105 – Funcionamiento en redes híbridas – Ping h1 a dispositivo móvil (IP 192.168.1.31)

7) Se accede a la interfaz web del controlador y se visualiza la topología de red.

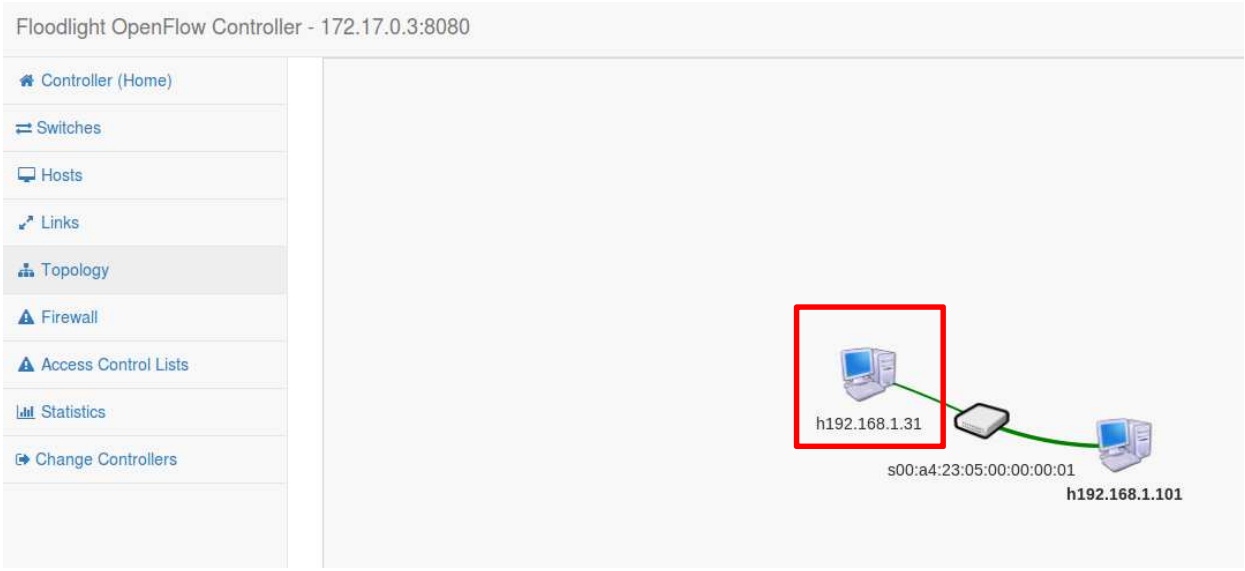


Figura Anexo 106 – Funcionamiento en redes híbridas – Topología de red vista desde el controlador. Se observa el dispositivo móvil.

Nota: La configuración IP en el host emulado por mininet, se realizó en forma manual. Dicha configuración se requiere debido a que el switch virtual por su configuración ve únicamente al host físico conectado a la red WiFi y no al servidor de DHCP.

Para que el host obtenga una dirección IP a través de dicho servicio, sería necesario realizar la configuración correspondiente desde OVX para que el switch virtual tenga una conexión hacia dicho servidor.

Esta configuración no es una limitación de la solución de virtualización, sino que se debe a la configuración que se realizó para la presente prueba.

La configuración de ip estática en el host h1 se realiza a través del siguiente comando:

mininet> h1 ifconfig h1-eth0 192.168.1.101 netmask 255.255.255.0

Tal como se aprecia en la Figura Anexo 106, el switch virtual ve a la tablet como un host conectado a través de un cable directamente con él.

8) Se procede a analizar los flujos en el switch:

mininet> dpctl dump-flows

*** s1 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x100000002, duration=463.306s, table=0, n_packets=462, n_bytes=45276, idle_timeout=5, idle_age=0, priority=1,ip,in_port=2,dl_src=f2:f4:ae:f5:58:1c,dl_dst=94:b1:0a:e1:68:4e,nw_src=192.168.1.101,nw_dst=192.168.1.31 actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,mod_nw_src:192.168.1.101,mod_nw_dst:192.168.1.31,output:1

cookie=0x100000001, duration=463.186s, table=0, n_packets=462, n_bytes=45276, idle_timeout=5, idle_age=0, priority=1,ip,in_port=1,dl_src=94:b1:0a:e1:68:4e,dl_dst=f2:f4:ae:f5:58:1c,nw_src=192.168.1.31,nw_dst=192.168.1.101 actions=mod_nw_dst:1.0.0.1,mod_nw_src:1.0.0.2,mod_nw_src:192.168.1.31,mod_nw_dst:192.168.1.101,output:2

Las reglas de flujo cargadas en el switch se corresponden con la topología de red que se aprecia a través del controlador.

Comunicación con la red emulada con GNS3

Desde el router configurado se logró realizar la comunicación con la Tablet, pero no así con la interfaz eth0 de la PC de prueba. Debido a ello, tampoco es posible realizar un ping al h1 (emulado a través de mininet). Se debe recordar que el h1 se conecta al switch y éste a la interfaz eth0.

Esto se debe a que GNS3 se ejecuta sobre Linux (Ubuntu 14.04 LTS) [75].

Esta condición de funcionamiento está asociada al diseño de GNS3 y no con su interacción con una red SDN virtualizada, con lo cual por lo expuesto [75] no se toma como input válido para la calificación de éste atributo.

D. FlowVisor: Virtualización de la red

Para la presente prueba se utilizó la misma configuración de red presentada en la Figura 32. La única excepción se presenta en que el docker correspondiente a OVX se reemplaza por el de FV. La configuración IP para este último se mantiene (IP 172.17.0.2). Se configura la red virtual de manera tal de que el host emulado a través de mininet pueda recibir IP dinámica a través del servicio de DHCP del proveedor de internet. En este caso no se configura el segmento de red emulado por GNS3.

A continuación, se expone la configuración realizada en la interfaz de administración de FV:

1) Creación del slice “virtual” y asignación al controlador floodlight (172.17.0.3)

```
# fvctl -f /dev/null add-slice virtual tcp:172.17.0.3:10000 admin@virtualslice
Slice password:
Slice virtual was successfully created
```

2) Creación de los “espacios de flujo” asignados al slice “virtual”.

```
# fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1 virtual=7
# fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=2 virtual=7
FlowSpace dpid1-port1 was added with request id 1.
```

Se configura la red virtual para que todo el tráfico que ingrese por el puerto 1 y 2 del switch emulado en mininet se administre a través de ella.

Se destaca nuevamente que al igual que en la experiencia realizada con OVX, la interfaz con la red física es el puerto 1 del switch OVS. Por su configuración a través de mininet está conectado (*bridged*) con la interfaz física de la PC.

3) Se corrobora la configuración de la red virtual en la administración de FV:

```
# fvctl -f /dev/null get-config
{
  "api_jetty_webserver_port": 8081,
  "api_webserver_port": 8080,
  "checkpointing": false,
  "config_name": "default",
  "db_version": "2",
  "enable-topo-ctrl": false,
  "flood-perm": {
    "dpid": "all",
    "slice-name": "fvadmin"
  },
  "flow-stats-cache": 30,
  "flowmod-limit": {
    "fvadmin": {
      "00:00:00:00:00:00:00:01": -1,
      "any": null
    },
    "virtual": {
      "00:00:00:00:00:00:00:01": -1,
      "any": null
    }
  },
  "host": "localhost",
  "log_facility": "LOG_LOCAL7",
```

"log_ident": "flowvisor",
"logging": "NOTE",
"stats-desc": false,
"track-flows": false,
"version": "flowvisor-1.4.0"
}

4) Se procede a realizar un ping entre el host1 emulado en mininet y la Tablet conectada a la red WiFi.

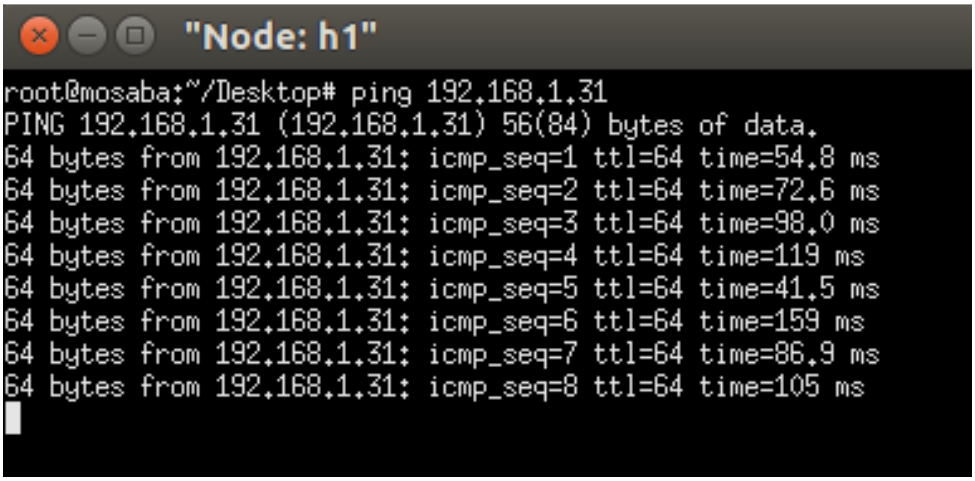


Figura Anexo 107 – Funcionamiento en redes híbridas – ping desde el host 1 a la Tablet (IP 192.168.1.31)

5) Se accede a la interfaz web del controlador y se visualiza la topología de red.

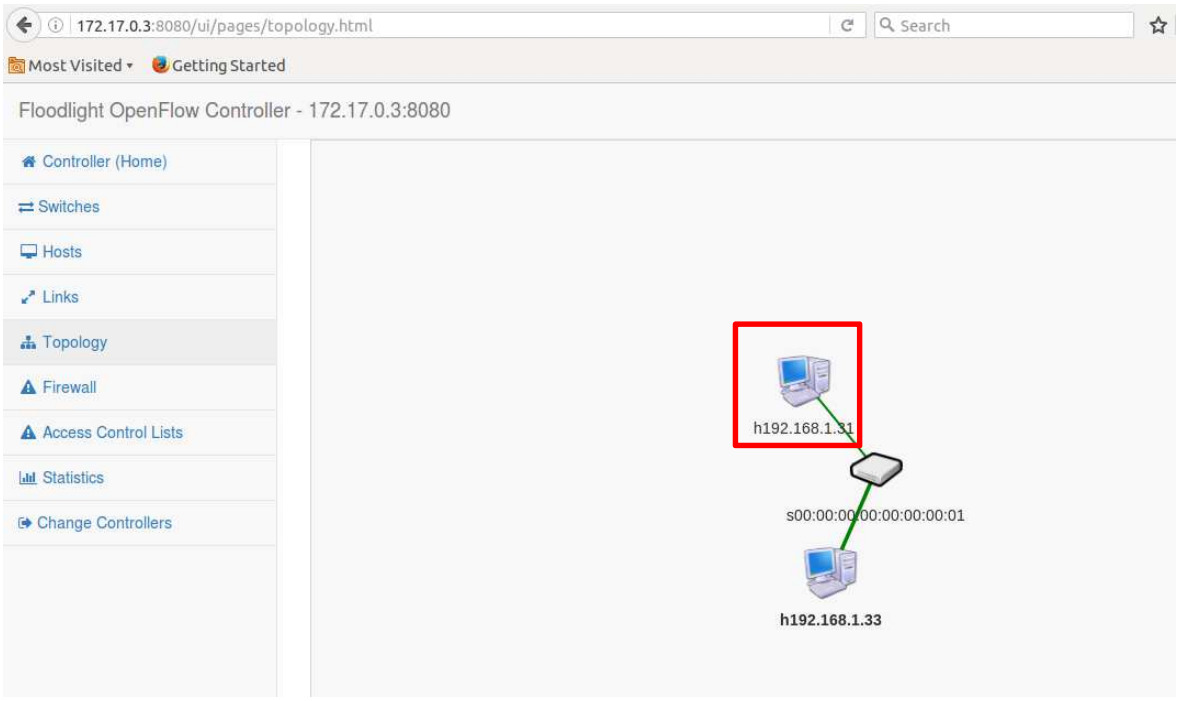


Figura Anexo 108 - Funcionamiento en redes híbridas – Topología de red vista desde el controlador. Se observa el dispositivo móvil.

6) Se procede a analizar los flujos en el switch:

*** s1 ----------

NXST_FLOW reply (xid=0x4):

cookie=0x133, duration=490.265s, table=0, n_packets=489, n_bytes=47922, idle_timeout=5, idle_age=1, priority=1,ip,in_port=2,dl_src=a6:b3:ad:59:cd:f6,dl_dst=94:b1:0a:e1:68:4e,nw_src=192.168.1.33,nw_dst=192.168.1.31 actions=output:1

cookie=0x134, duration=490.215s, table=0, n_packets=487, n_bytes=47726, idle_timeout=5, idle_age=1, priority=1,ip,in_port=1,dl_src=94:b1:0a:e1:68:4e,dl_dst=a6:b3:ad:59:cd:f6,nw_src=192.168.1.31,nw_dst=192.168.1.33 actions=output:2

Siendo 94:b1:0a:e1:68:4e la dirección MAC de la Tablet y dl_src=a6:b3:ad:59:cd:f6 la dirección MAC asignada al host emulado en mininet.

Las reglas de flujo cargadas en el switch se corresponden con la topología de red que se aprecia a través del controlador.

NOTA: la red virtual se configuró para que todo el tráfico que ingresa por el puerto 1 sea administrado por la red virtual. Por ello en este caso es posible la configuración de IP en el host a través del servicio de DHCP del proveedor de servicio de internet.

Para ello se ejecuta el siguiente comando en la consola del h1.

```
# dhclient h1-eth0
```

La dirección IP asignada en este caso es la que se aprecia en la figura Figura Anexo 109.

La dirección MAC cambió respecto a la experiencia con OVX, ya que ambas simulaciones se realizaron en tiempos distintos y fue necesario ejecutar el script que instancia la red emulada en mininet en dos oportunidades. Las direcciones MAC de los hosts en dicho script no están explicitadas (*hardcoded*). Debido a ello, cada vez que se ejecuta nueveamente, se asigna una dirección MAC distinta al host (Figura Anexo 109).

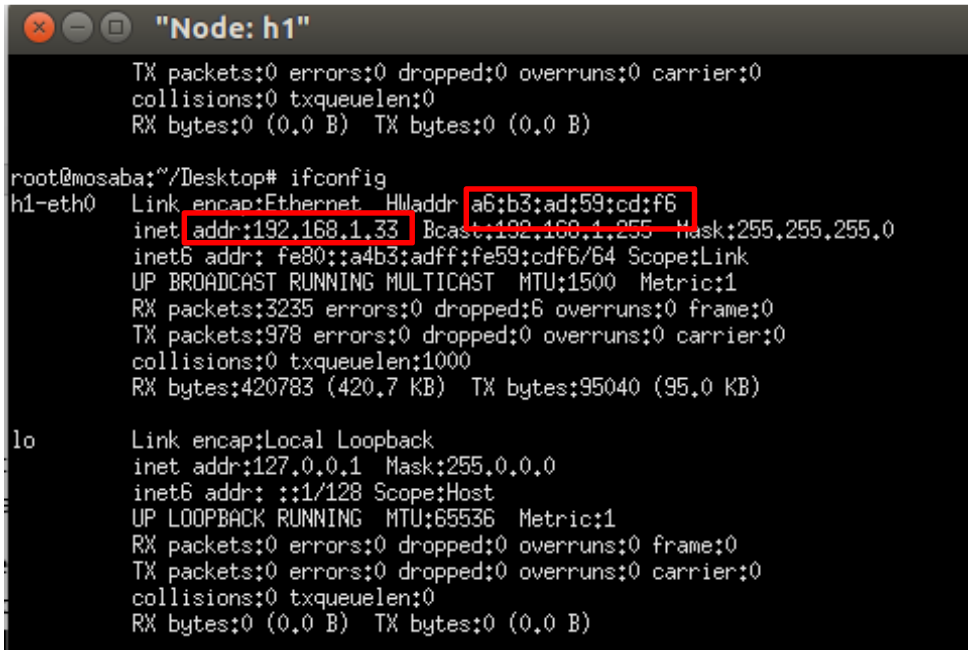


Figura Anexo 109 – Funcionamiento en redes híbridas – Direcciones MAC e IP del host 1