SEDE GUAYAQUIL

FACULTAD DE INGENIERÍAS

CARRERA DE INGENIERÍA ELECTRÓNICA

Proyecto de Graduación

Previo a la obtención del título de:

INGENIERO ELECTRÓNICO

TÍTULO:

DISEÑO Y SIMULACIÓN DE UNA RED DE DATACENTERS BASADA EN TOPOLOGÍA FAT-TREE EN UN AMBIENTE DE REDES DEFINIDAS POR SOFTWARE (SDN).

AUTORES:

-Alan Roberto Ampuño Avilés

-Mayra Michell Chávez Cristóbal

DIRECTOR:

MSc. Javier Ortiz Rojas

GUAYAQUIL-ECUADOR

2015

Declaración de responsabilidad

Nosotros Alan Roberto Ampuño Avilés y Mayra Michell Chávez Cristóbal autorizamos a la Universidad Politécnica Salesiana la publicación total o parcial de este trabajo de grado y reproducción sin fines de lucro.

Además declaramos que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

_____ _____

Alan Roberto Ampuño Avilés
CC 0930114343

Mayra Michell Chávez Cristóbal
CC 0930296363

Dedicatoria

Dedico este trabajo a Dios, por permitirme cumplir con este objetivo.

A mis padres y mis hermanos por guiarme y enseñarme que el camino a seguir es siempre el del estudio y el esfuerzo para lograr todos los sueños que tenga.

Alan Ampuño A.

Dedicatoria

A Dios por ser la guía en todo momento y sólo gracias a su amor he podido llegar hasta este momento.

A mi familia por su apoyo en cada etapa, por enseñarme que con esfuerzo cualquier propósito se puede lograr y por inculcarme siempre que los sueños no tienen límites.

En especial dedico este trabajo a mi hermana Johanna, por creer siempre en mí y por la bendición que es tenerla a mi lado como hermana y mejor amiga.

Mayra Chavez C.

Agradecimiento

Agradezco a Dios, por darme la salud y el intelecto para alcanzar este objetivo.

A mis padres por cuidarme e inculcarme con amor y paciencia los valores y principios para ser un hombre de bien.

A mis hermanos Fabián, Gary y Andrea por mostrarme con su ejemplo cual era el objetivo al que debía apuntar.

Al Ing. Javier Ortiz quien me recibió en el Departamento de Sistemas hace ya 8 años como pasante, jefe, amigo y finalmente como tutor de tesis para ser la guía de este trabajo.

A todos los docentes y en especial al Ing. Victor Huilcapi por todo el apoyo brindado en la carrera.

A Mayra Chavez por ofrecerme su confianza y acompañarme en este largo camino.

Alan Ampuño A.

Agradecimiento

Agradezco a Dios, sé que sin Él nada hubiera sido posible, por regalarme la oportunidad de vivir con un propósito

A mi familia por cuidar de mí, por todo su amoroso apoyo en cada momento.

A mi hermana Johanna por ser mi sabia consejera, por todo el ánimo y la ayuda que me brinda siempre, definitivamente no lo hubiera logrado sin su compañía.

A todos los educadores que me inculcaron el amor a la carrera, sobre todo al Ing. Victor Huilcapi, nuestro director por su excelente trabajo y aporte, a mi tutor, el Ing. Javier Ortiz por todo su apoyo a lo largo del desarrollo del proyecto.

A todos mis amigos por estar siempre dispuestos a ayudar, en especial a mi mejor amigo y compañero de tesis Alan, por su infinita paciencia y trabajo, por el privilegio que tengo de haberlo conocido.

Mayra Chavez C.

Índice General

Índice de Tablas

Índice de Gráficos

Resumen

AÑO	ALUMNOS	DIRECTOR DE TRABAJO DE TITULACIÓN	TEMA TRABAJO DE TITULACIÓN
2014	ALAN ROBERTO AMPUÑO AVILÉS MAYRA MICHELL CHAVEZ CRISTOBAL	ING. JAVIER ORTIZ	DISEÑO Y SIMULACIÓN DE UNA RED DE DATACENTERS BASADA EN TOPOLOGÍA FAT-TREE EN UN AMBIENTE DE REDES DEFINIDAS POR SOFTWARE (SDN)

Debido al aumento en la generación de tráfico originado en los Centros de Datos (DC) ocasionado por el creciente aumento de usuarios con acceso a Internet, se hace inherente la aparición de problemas de congestión, lo que provoca que el DC tenga que reestructurar su infraestructura para solventar los inconvenientes y así mantener la calidad de los servicios proporcionados. El establecer un escenario simulado por computadora, que contribuya al estudio de las problemáticas existentes en estos DC, es el objetivo del presente trabajo, el cual trata sobre el diseño y la implementación de una red de DC basada en topologías Fat-Tree, bajo un ambiente de Redes Definidas por Software.

Los programas que formaron parte de este proyecto incluyen el uso de un Controlador de SDN OpenDayLight, el cual se encargó de administrar la red que fue simulada mediante el software Mininet. La generación de tráfico con comportamiento estadístico se realizó a través del software DITG.

Para la comprobación de este escenario se escogieron métricas de red basadas en invariantes, y cuyos resultados teóricos mantuvieron similitud con lo obtenido a través de la experimentación que fue realizada en condiciones aproximadas al de un DC real.

A futuro este escenario puede servir como una herramienta que permita a los investigadores ensayar nuevas propuestas de solución sobre problemáticas en DC con otras topologías o tamaños de red.

PALABRAS CLAVES

Redes Definidas por Software, Controlador, Opendaylight, Mininet, Fat-Tree, Topologías, Generador de Tráfico, DITG, Métricas de red, Invariantes, Redes de Datacenters.

.

Abstract

YEAR	AUTHORS	ADVISOR	TITLE
2014	ALAN ROBERTO AMPUÑO AVILÉS MAYRA MICHELL CHAVEZ CRISTOBAL	ING. JAVIER ORTIZ	DESIGN AND SIMULATION OF A DATACENTERS NETWORKS BASED IN FAT-TREE TOPOLOGY IN SOFTWARE DEFINED NETWORK (SDN) ENVIROMENT

Due to the recent increase of internet users, Internet traffic caused by Data Centers have grown, leading to Internet congestion problems, this brings about major changes in the Data Centers Infrastructure in order to overcome drawbacks and offer good quality services.

The objective of this proposal is the design and implementation of a DC network based in Fat-Tree topologies under a Software Defined Networking environment, by establishing a computer-simulated scenario to contribute with the study of existing problems in these DC

The Software involved in this project incorporate the use of Open Daylight, a SDN controller was in charge of running the Network Administration, simulated by Mininet software. The D-ITG software was the responsible for generating traffic with statistical behavior.

In order to test this scenario, network metrics based on invariants were chosen and the theoretical results obtained keep similarity with the results obtained through the experimentation in approximate conditions to a real DC.

This project lays the groundwork for future researches and can be used as a tool for testing new DC solutions proposals with different topologies or network sizes.

KEYWORDS

Software Defined Network, Controller, Opendaylight, Mininet, Fat-Tree, Topologies, Traffic Generator, DITG, Networks Metrics, Invariants, Datacenters Networks,

Introducción

El proyecto propone el diseño y la implementación de una red de Datacenters basada en topología Fat-Tree en un ambiente de redes definidas por software.

Se plantea principalmente la generación de un escenario fiable y con un comportamiento cercano a la realidad que contribuya al estudio de problemáticas presentadas en redes de Datacenters, éste podrá ser utilizado por investigadores y/o estudiantes que no cuenten con facilidades de acceso a grandes Centros de Datos[1].

El uso de las herramientas de simulación permite modelar situaciones, para este caso se utiliza el software Mininet para la creación de la topología; OpenDayLight para levantar un controlador de redes definidas por software y para la generación del tráfico estadístico DITG.

El procedimiento que corresponde a la validación se realizará a través del uso de métricas de red basadas en invariantes.

En el capítulo uno se detalla la introducción hacia el problema y la propuesta que plantea este trabajo, así como el objetivo, beneficiarios e impacto del mismo.

El capítulo dos contiene información relevante sobre cada una de las herramientas utilizadas y su concepto científico, pretende ampliar el conocimiento sobre el tema a tratar, explicando aspectos importantes sobre SDN y la función del Controlador, también se exponen los criterios y características que presentan las topologías aplicadas en Datacenters.

El capítulo tres muestra las características de lo planteado, se describen los componentes a utilizar para la elaboración el escenario propuesto, detalla la forma en cómo se realizará la validación del escenario y el aspecto teórico sobre las métricas de red basadas en invariantes con las que se analizará el comportamiento de la red simulada.

En el capítulo cuatro se exponen los resultados de lo obtenido a partir de la experimentación de la topología de red en la herramienta de simulación en contraste con los valores de las métricas de red escogidas.

[1] DC o Centro de Datos

Capítulo I

El problema

1.1 Planteamiento del problema

Actualmente el incremento de usuarios con acceso a Internet ha causado un aumento significativo del tráfico que se genera en un Datacenter, lo que causa la aparición de problemas de congestión, bajo rendimiento de las aplicaciones, alta tasa de errores y pérdidas de paquetes, entre otros. El Datacenter debe ser capaz de garantizar la calidad del servicio en todo momento haciendo necesaria la tarea de modificar la infraestructura actual con el objetivo de solventar las nuevas exigencias. Este proceso implica en muchas ocasiones que se detengan los servicios durante el tiempo que se apliquen los cambios.

Por otro lado para el estudio de soluciones a menudo se requiere el acceso a la red de los Datacenters, lo cual muchas veces resulta ser una tarea compleja, ya que esto representa un riesgo elevado para los servicios que se están proporcionando.

Tomando en cuenta la problemática presentada, la aparición de Redes Definidas por Software plantea una ventaja interesante: permite otorgar en un solo equipo toda la administración de la red, haciéndola capaz de soportar cambios en toda la estructura sin necesidad de detener los servicios.

Para el estudio de este tipo de redes es frecuente el uso de herramientas de simulación, ya que plantean un escenario de red que cuente con topologías y flujos de tráfico con características similares a un Datacenter.

1.2 Delimitación del problema

El presente proyecto está dirigido a los investigadores con nuevas propuestas de estudio para redes de Datacenters y esperan poder contar con un escenario donde se pueda analizar problemáticas basadas en Fat-Tree.

Este trabajo se implementó en un servidor del área de Sistemas de la Universidad Politécnica Salesiana cuyas prestaciones técnicas permitieron preparar una máquina

virtual con el software de simulación de redes Mininet. Se útilizó OpenDayLight como controlador SDN, y D-ITG como herramienta para emular tráfico estadístico. La topología elegida fue Fat-Tree debido a que se encuentra entre las más sencillas de implementar en casos de estudio de Centros de Datos.

Este proyecto se limita a los conocimientos adquiridos en la carrera de Ingeniería Electrónica Mención Telecomunicaciones, orientándose a las materias de redes de computadoras 2 y 3.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar y simular una red de Datacenters basada en topología Fat-Tree en un ambiente de Redes Definidas por Software para proporcionar un escenario adecuado que permita el estudio de soluciones a los problemas de red bajo gestión de SDN

1.3.2 Objetivos específicos

- Exponer las características principales de la topología más sencillas de implementar en casos de estudio y Data Centers: Fat-Tree.
- Utilizar el software Mininet como herramienta de simulación para entornos de redes definidas por software.
- Emular a Fat-Tree con Mininet y generar tráfico con D-ITG.
- Validar el escenario planteado utilizando invariantes de red para grandes centros de datos utilizando topología Fat-Tree.
- Exponer el análisis de los invariantes de red propuestos en la topología Fat-Tree emulada bajo Mininet.

1.4 Justificación

El presente estudio posee un valor teórico con aportación a la investigación en este campo.

Las plataformas de simulación/emulación, son de gran utilidad ya que ayudan a implementar un ambiente de red y trabajar en él, permitiendo la experimentación de propuestas científicas que se aplicarían en escenarios reales.

Se considera que este escenario conceda a los futuros investigadores un punto de partida para el desarrollo de ideas hacia nuevas implementaciones, como por ejemplo, su uso en otras topologías o con diferentes controladores SDN.

1.5 Variables e indicadores

El escenario planteado en este proyecto se utilizan como variables dos métricas de red para comparar los resultados experimentales con los analíticos y así determinar su validez, estás se detallan en la Tabla 1.

Tabla 1
Variables e indicadores

VARIABLES	INDICADORES
Diámetro de la red	Longitud máxima del camino más corto entre todo par de hosts de la red. Se mide en números de saltos.
Longitud de ruta promedio	Esfuerzo promedio que realizan todos los paquetes durante el desarrollo normal del tráfico, Se mide en números adimensionales

Nota: Se describen las variables e indicadores a utilizar en el proyecto. Por: Los autores

1.6 Metodología

1.6.1 Métodos

El presente trabajo de investigación hace referencia a los siguientes métodos:

Método Experimental

Se utilizó el método experimental, al realizar pruebas de generación de tráfico en el escenario planteado y así obtener resultados que se analizaron a través de las métricas de red.

Método Deductivo

Las herramientas científicas que se usaron como la implementación de la topología en Mininet y el posterior análisis del comportamiento de la red con tráfico generado

permitieron obtener un nuevo resultado que aporta a las futuras investigaciones en este campo.

1.6.2 Técnicas

Observación

La observación consistió en registrar el comportamiento de la red en cada una de las pruebas en donde se generó tráfico estadístico y estos eventos se analizaron posteriormente con las métricas de red basadas en invariantes.

1.6.3 Instrumentos de Investigación y Recolección de Datos

Los instrumentos de la investigación en los cuales se pudo obtener información relevante para esta investigación fueron:

* Libros

* Blogs

* Papers

* Cursos Online

* Videos Online

Para analizar y capturar todo el tráfico que pasa a través de la red y su posterior validación, se utilizó el software Wireshark[2].

1.7 Población y muestra

Los Centros de Datos que utilicen Fat-Tree como topología de red serán tomados como población y la muestra corresponde a los Centros de Datos con implementación Fat-tree de acuerdo al siguiente esquema: 4 switches correspondientes al nivel de núcleo, 8 switches en el nivel de distribución y 8 switches para el nivel de acceso, bajo esta implementación se pueden colocar en la red 16 clientes o servidores.

[2] Wireshark es un analizador de protocolos con interfaz gráfica que permite capturar información que pasa sobre una red

1.8 Descripción de la propuesta

Se propone establecer un escenario para el desarrollo de investigaciones sobre topologías de Datacenters, en un ambiente de redes definidas por software. En este caso se va a simular una topología Fat-Tree, debido a la facilidad para su implementación en casos de estudio que involucren Centros de Datos.

Se usará el software Mininet dentro de una máquina virtual de Ubuntu con la ayuda de Virtual Box, donde a través de un script basado en *bash*[3], se emulará la topología Fat-Tree (k=4), generando tráfico con comportamiento estadístico mediante la herramienta D-ITG que sea semejante a un tráfico real.

La validación de este escenario se hará por medio de conocidas métricas de red basadas en invariantes, de donde se obtendrán resultados teóricos que serán comprobados de acuerdo a los resultados experimentales obtenidos a través de la simulación y posterior observación del comportamiento según tráfico generado, esperando que estos valores sean de gran aproximación con lo mostrado teóricamente.

1.8.1 Beneficiarios

Este proyecto pretende aportar un escenario adecuado para todos los investigadores que deseen contribuir al estudio de las temáticas presentadas en las redes de Datacenters basadas en redes definidas por software.

Este documento además contiene el procedimiento detallado para la instalación y ejecución de cada una de las herramientas de simulación lo que será de gran ayuda para quienes aspiren ampliar este trabajo en futuras tesis.

1.8.2 Impacto

Este trabajo muestra la factibilidad de usar herramientas de simulación, que bajo las correctas condiciones, permiten crear escenarios de red que se aproximan a lo real, y que contribuirá al desarrollo de nuevas propuestas de solución en este campo, partiendo de lo que aquí se plantea, y dando continuidad a esta investigación.

[3] Bourne Again Shell

Las implicaciones de este proyecto son de tipo académico científico, pues su ejecución aporta a la realización de escenarios experimentales basados en Redes Definidas por Software, una nueva tecnología que plantea una propuesta interesante que merece ser analizada, puesto que concentra toda la administración de la red en un solo equipo o Controlador; además de la posibilidad de poder realizar cambios en la estructura sin tener que detener los servicios.

Capítulo II

Marco Teórico

2.1 Antecedentes

La evolución de las redes permitió ofrecer una amplia variedad de servicios, todo este logro trajo consigo el aumento en la complejidad de la red haciendo de su administración una tarea difícil. Surge entonces la necesidad de utilizar los recursos de manera óptima haciendo la infraestructura lo más eficiente posible capaz de soportar el aumento de tráfico generado.

Lograr una red manejable también ha sido un reto que alcanzar por lo que los avances de investigación para obtener un control eficaz y dinámico de la red han estado presentes en todo momento.

2.2 Topologías

(Liu, Muppala, Veeraghavan, Lin, & Hamdi, 2013) Define como topología de red a los diferentes tipos de estructura en que se organiza una red para efectuar la transmisión de información entre los dispositivos.

Cada topología que se lleve a cabo implica tener incorporada una topología física y una topología lógica. La topología física, especifica la estructura de los equipos, por ejemplo, cómo deben estar conectados los cables entre los dispositivos. La topología lógica por otro lado, define el conjunto de normas que están asociadas a la topología física, como el modo en que se debe administrar la transmisión de la información (Liu et al., 2013).

La infraestructura del Datacenter, ha ocupado un importante interés en recientes estudios debido a la acelerada evolución de las aplicaciones implementadas dentro del mismo. Se requiere que sea escalable, fácil de administrar y que posea tolerancia a fallos (Liu et al., 2013).

2.2.1 Clasificación de las topologías

A continuación se muestra la clasificación de las arquitecturas de red para un Datacenter:

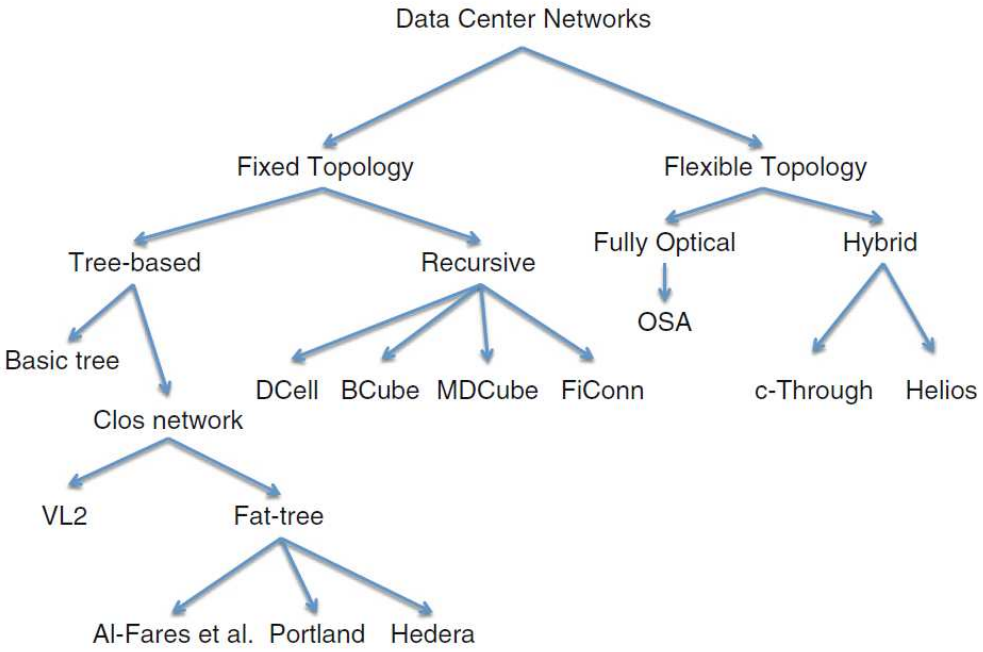

Figura 1 Taxonomía de una topología de Datacenter. Por: (Liu, et al., 2013, p. 16)

Se han clasificado de acuerdo a la habilidad que poseen para reconfigurarse mientras están en ejecución.

2.2.1.1 Fixed Topology Tree Based

Es la más comúnmente usada en los Centros de Datos.

Basic Tree

Consiste en dos o tres niveles de switches/routers, y no existen conexiones entre equipos en el mismo nivel. El número de servidores en esta topología es limitado por el número de puertos de los switches. (Liu et al., 2013)

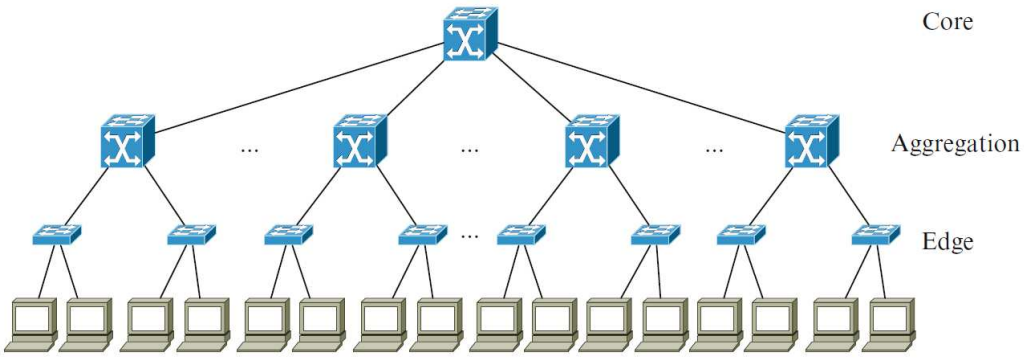

Figura 2 Topología Basic-Tree. Por: (Liu et al., 2013, p. 17)

Topología Fat-Tree

Su diseño es jerárquico y cuenta con n capas claramente definidas como son: switches de acceso *(Edge)* que se conectan a los switches de distribución *(Aggregation*), y estos a su vez se comunican con los switches de núcleo *(Core)*. (Liu et al., 2013)

Estructura

En una topología de Data Center, para cada k puerto de un switch en la capa de acceso, $\frac{k}{2}$ se encuentran conectados a los servidores. Los restantes $\frac{k}{2}$ puertos, se conectan a $\frac{k}{2}$ switches en la capa de distribución. Estas conexiones forman una célula básica de Fat-tree a la que se conoce como ***pod***[4]. En el núcleo, existen $\left(\frac{k}{2}\right)^2$ n puertos de switch, cada uno conectado hacia n pod.

El máximo número de servidores o hosts conectados en una topología Fat-tree con n puertos de switch es $\frac{k^3}{4}$. (Xavier , 2012)

[4] Pod, nodo.

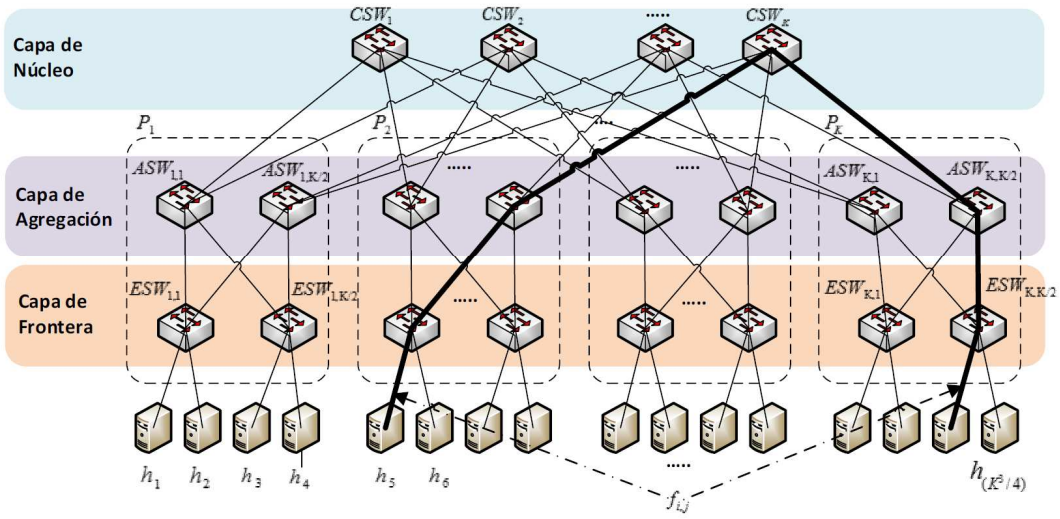

Figura 3 Topología Fat-Tree. Por: (Ortiz, Ampuño, Chávez, Londoño, & Novillo, 2014)

2.2 Redes Definidas por Software

2.2.1 Antecedentes

Parte de la evolución de las telecomunicaciones en general, es poder llegar a tener el control absoluto de todo el esquema de la red de una manera simplificada que permita al administrador de la red manejar una topología flexible y escalable sin que esto signifique grandes esfuerzos y este ha sido un proceso que se ha ido desarrollando desde hace ya mucho tiempo. En los años 80´s se logró plantear un diseño centralizado para el control de la red, luego en la década de los 90´s se introdujo la programación de redes, así como también el concepto de virtualización.

2.2.2 Definición

Según lo expresado por (Astudillo A., Dreier G., & Tarouco L., 2012):

> Una red definida por software consiste básicamente en entregar el control y la inteligencia de los dispositivos de la red a un solo servidor externo o controlador. Este servidor se encargará de manejar el tráfico a través de entradas que son las aplicaciones de los usuarios que le dirán a los equipos qué camino tomar seleccionando la mejor ruta constantemente. Esto permite tener una visión y control global de la red

La estructura de la red no presenta una configuración fija o determinante, sino más bien flexible, de acuerdo al comportamiento de las aplicaciones, se entrega el control al usuario y el uso que éste haga de la red, entonces se dice que de acuerdo a los diferentes factores como calidad del servicio, tráfico, número de requerimientos o tipo de aplicación, el Controller es capaz de tomar diferentes decisiones que aseguren la entrega efectiva de información eligiendo cada vez el mejor camino.

Las SDN fueron creadas para funcionar en la era de la nube, producen cambios importantes en cómo se construyen las redes ya que desprende el concepto de control con hardware, y éste se le otorga a una aplicación de software o Controlador.

2.2.3 Características

De acuerdo a lo expuesto por (Linux Foundation, 2014) las características principales que se destacan en SDN son las siguientes:

- Optimiza el diseño de la red y la utilización de equipos.
- Posibilita el desarrollo acelerado de nuevas capacidades de red ya que no es necesario pensar en que los cambios se deben aplicar en cada equipo de la topología.
- Se obtiene una red que se adapta a las necesidades del usuario, según lo que se requiera en el momento, mejorando la experiencia del usuario.
- Convierte una red estática en una flexible y con más inteligencia que permite, por ejemplo, asignar recursos dinámicamente.
- Mejora la escalabilidad, administración y agilidad de la red.
- Al tener una inteligencia centralizada es posible alterar, en tiempo real, el comportamiento de la red y desplegar servicios y aplicaciones nuevos en cuestión de horas o días en vez de meses

2.2.4 Arquitectura

Su estructura está compuesta básicamente de tres planos bien diferenciados:

- **Capa de infraestructura**, se refiere al hardware, sus componentes son los dispositivos de la red, no presentan mayor inteligencia por lo que son equipos de bajo coste, sin necesidad que sean del mismo fabricante, el único requisito

es que cuenten con soporte para *OpenFlow*[5] que es el protocolo que permite la comunicación entre los dispositivos y el Controller.

- **Capa de control**, es aquí donde se encuentra el Controller, quien maneja toda la inteligencia de la red, se comunican con la capa de infraestructura a través de OpenFlow y posee una API[6] abierta para que pueda ser administrado por la capa de aplicación.
- **Capa de aplicación**, de acuerdo a este esquema es el nivel más alto desde donde se puede controlar el comportamiento de la red, a través de la API del controlador. (Rojas A. & Pachón , 2013)

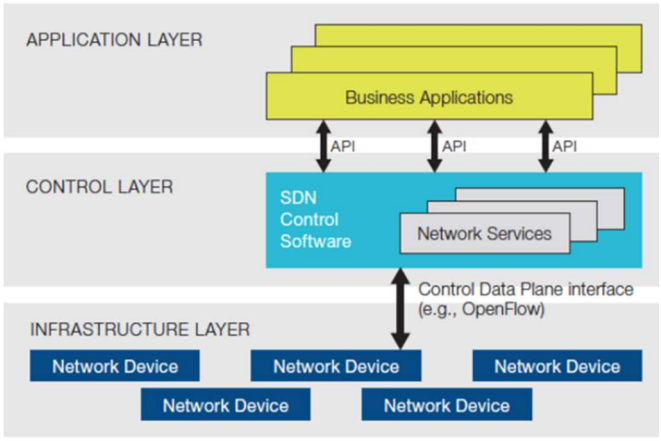

Figura 4 Arquitectura SDN. Por: (Rojas A. & Pachón, 2013)

2.2.5 Controlador SDN

Refiriéndonos nuevamente a un concepto de la red, se dice que:

> El Controlador centraliza toda la comunicación que pasan por los elementos de red, funciona como un sistema operativo y tiene una visión general de toda la red. Las aplicaciones que se ejecutan en el controlador determinan cómo los flujos van a comportarse por la misma.
>
> Cada elemento de la red (*switches*, AP) se reportan con el controlador siempre que sea necesario y éste habla OpenFlow con los equipos (Astudillo et al., 2012).

[5] Estándar abierto que permite a los investigadores ejecutar protocolos experimentales en el campo de las redes.
[6] API (Aplication Programming Interface), Interfaz de Programación de Aplicaciones, provee la capacidad de comunicación entre componentes de software.

Todos los elementos de la red se comunican con el controlador y viceversa a través de OpenFlow. Una aplicación de usuario, permitirá al usuario controlar y administrar la red. En el mercado actualmente existen varios controladores, que varían según la implementación que se requiera, y se clasifican de acuerdo al programa fuente en que están basados, entre ellos:

- *Python*[7]
- C y C++
- Java
- JavaScript
- Ruby

De acuerdo a esta clasificación se disponen de varios ejemplos de controladores según su plataforma como se detalla en la Tabla 2.

Tabla 2
Clasificación de controladores

CONTROLADOR / PLATAFORMA	PYTHON	C	C++	JAVA	JAVASCRIPT	RUBY
POX						
MUL						
NOX						
JAXON						
TREMA						
BEACON						
FLOODLIGHT						
MAESTRO						
RYU						
NODEFLOW						
CONTROLLER						
OPENDAYLIGHT						

Nota: Se detallan las plataformas que utilizan algunos controladores. Por: Los autores

2.2.6 Openflow

Según lo indica (Spera, 2013) :

[7] Lenguaje de Programación basado en scripts orientado a objetos.

OpenFlow es el primer estándar que se ha definido para la comunicación entre las capas de arquitectura y control en una arquitectura SDN. Permite el acceso directo y manipulación del plano de enrutamiento de los equipos de la red, de una manera virtual, se podría decir que es capaz de mover el control de la red desde cada uno de los equipos hacia uno solo.

OpenFlow usa el concepto de "flujos", para identificar tráfico de la red basado en reglas pre-definidas que pueden ser así mismo reprogramadas por el software de control SDN. Permite también a las *IT*[8], definir como el tráfico debe atravesar la red, basado en diversos parámetros, como los patrones de uso, tipo de aplicaciones o recursos de la nube que pueden ser modificados en tiempo real, algo con lo que no se cuenta en las arquitecturas de hoy en día. Este estándar puede ser desarrollado en las redes existentes, física y virtualmente. Los dispositivos de la red, deben poder soportar el protocolo.

2.3 Herramientas para la simulación

2.3.1 Controlador Opendaylight

OpenDayLight es un controlador modular, escalable, multiprotocolo, construido para SDN, capaz de integrarse a redes heterogéneas compuestas por varios fabricantes. El modelo impulsado, provee una capa de Abstracción de Servicio (SAL)[9] que permite el soporte de múltiple protocolos de comunicación hacia las capas inferiores (*Southbound*), a través de Plugins[10], y un set de API´s para las capas superiores (*Northbound*). (OpenDaylight, 2014)

2.3.1.1 Características

- Marco Basado en OSGI[11]
- Clustering
- Service Abstraction Layer (SAL)
- OpenFlow 1.0 Southbound Plugin

[8] IT (Information Technology), Tecnologías de la Información
[9] SAL (Service Abstraction Layer)
[10] Aplicación cuya función es aportar una función adicional y específica a otra aplicación. Llamado también extensión
[11] Servicios para aplicaciones co-localizadas

- Northbound API (OSGi Framework and REST)
- Host-Tracker
- Enrutamiento basado en Dijkstra
- Estadísticas
- Interfaz GUI
- Administración de Enrutamiento
- Administración de Switches
- Administración de la topología
- Aplicaciones de ejemplo (OpenDaylight, 2014).

2.3.1.2 Arquitectura

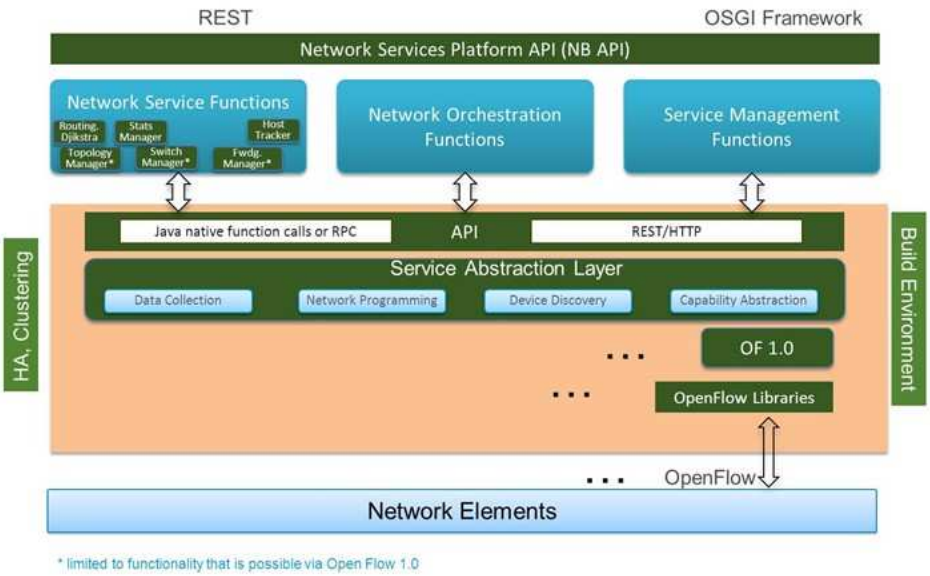

Figura 5 Arquitectura OpenDayLight. Por: (Wiki OpendayLight, 2014)

2.3.1.3 Service-Abstraction-Layer

Es el corazón del diseño modular para el Controlador y permite soportar múltiples protocolos para la comunicación Southbound, provee un servicio consistente para módulos y aplicaciones (donde el control lógico de operaciones está concentrado). SAL proporciona servicios básicos, como el descubrimiento de la red, que son usados por otros módulos, como el de Administración de la Topología para construir el plano de la red y las capacidades de cada dispositivo.

Los servicios se construyen usando las características expuestas por el Plugin, de esta forma permite usar el protocolo Southbound más apropiado para interactuar con el dispositivo descubierto. Otros servicios que proporciona son:

- El servicio de topología permite transmitir información como la integración de un nuevo nodo o una nueva conexión que haya sido descubierta y activada.
- El servicio de Paquete de datos hace posible la entrega de los paquetes hacia la aplicación
- La programación de flujos permite la modificación y configuración de las tablas de flujos
- El servicio de Estadísticas da a conocer el resumen de los flujos, puertos de los nodos.
- El servicio de Inventario recolecta información acerca de los nodos y sus puertos para los API´s y
- El servicio de Recursos, presenta información de ayuda.

Este controlador es software puro y puede correr en cualquier sistema operativo que soporte también *Java*[12] (Wiki OpendayLight, 2014)

2.3.2 Mininet

Mininet es una plataforma que funciona como emulador de red. Permite crear dispositivos de red: host, switches, controladores y sus respectivas conexiones en una sola máquina virtual o física. Se puede ejecutar en una variedad de dispositivos finales, switches, routers que posean *Linux Kernel*[13]. Su virtualización es ligera y funciona correctamente con una red simple hasta con una red completa con el mismo Kernel, Sistema Operativo y código.

[12] Lenguaje de programación muy completo orientado a objetos, robusto, de arquitectura neutra, multitarea y dinámico.
[13] Núcleo de Linux, se denomina núcleo a la parte a la que se tiene acceso solo en modo privilegiado.

Un host Mininet se comporta como una máquina real, que contiene toda la información del sistema operativo, se puede ingresar por *SSH*[14] , ejecutar programas y enviar paquetes en lo que emula una interfaz Ethernet real, configurada con velocidad y retardo determinados. Los paquetes son procesados por el Switch o Router Ethernet virtual con una cantidad de "cola" predeterminada. (Lantz, Handigol, Heller, & Jeyakumar, 2014)

2.3.2.1 Características de Mininet

- Es rápido, empezar una red simple sólo toma unos pocos segundos, es decir que se puede editar, y correr nuevamente una red de una forma muy fácil y veloz.
- Se puede crear topologías personalizadas.
- Es posible correr programas reales, todo lo que se pueda ejecutar en Linux, como Servidores Web, herramientas de monitoreo como *Wireshark*
- Personaliza el reenvío de paquetes: los switches de Mininet son programables gracias al protocolo OpenFlow.
- Mininet puede ejecutarse en una pc, laptop, servidor, máquina virtual sin problemas.
- Se puede usar fácilmente, ejecutando experimentos de Mininet o creando un script simple o complejo basado en Python.
- Mininet es un proyecto de código abierto, es decir que se puede examinar y modificar sin restricciones.
- Se encuentra en constante desarrollo, así que si se encuentra algún inconveniente, duda o error, se puede compartir en **mininet-discuss** y la comunidad de desarrolladores estarán prestos a explicar, ayudar o reparar, colaborando así permanentemente. Asimismo cualquiera que se encuentre trabajando con esta herramienta está alentado a compartir algún parche para al menos informar sobre el error. (Lantz et al., 2014).

[14] SSH (Secure Shell). Intérprete de órdenes segura, protocolo de la capa de aplicación que permite acceso remoto.

2.3.2.2 Limitaciones de Mininet

A pesar de todos los atributos que posee esta plataforma, se presentan algunas limitaciones como son:

- Ejecutarse en un solo equipo es muy conveniente pero claramente impone límites en cuanto a los recursos de este equipo, se requiere al menos 3Gbps para tráfico simulado.
- Mininet no es capaz de configurar el Controlador OpenFlow por sí solo, si se requiere modificar flujos de rutas o el comportamiento de los switches, se necesita desarrollar un Controlador con las características que se requieran para la red.
- No realiza *NAT*[15], es decir que los hosts de Mininet se encuentran aislados de la red LAN por defecto, esto generalmente es algo bueno, pero significa que los hosts no pueden comunicarse directamente con el Internet a menos que se provean los medios necesarios para lograrlo, existen ejemplos demostrativos como nat.py (muestra como configurar un hosts Mininet para que tenga salida externa) y hwintf.py (demuestra como agregar una interfaz física a Mininet).
- Todos los hosts Mininet comparten los archivos del sistema, esto significa que se debe ser cuidadoso al manipularlos así mismo de no matar los procesos errados, (bind.py indica como tener directorios privados por host).
- A diferencia de un simulador, Mininet no presenta notoria noción del tiempo virtual, esto significa que las medidas de tiempo serán referidas al tiempo real, así que los resultados faster-than-real-time (100Gbps) no pueden ser fácilmente emulados (Lantz et al., 2014).

2.3.3 D-ITG

Distributed Internet Traffic Generator, es una plataforma capaz de producir tráfico ipv4 e ipv6 con bastante precisión como el que comúnmente se observa en las

[15] NAT (Network Address Translation), Traductor de Direcciones de Red, usado para lograr el intercambio de información entre direcciones IP incompatibles.

aplicaciones de internet. Al mismo tiempo D-ITG es también es una herramienta de medición con la que es posible medir el rendimiento de la red a partir de las métricas comúnmente conocida (*throughput*, *delay*, *jitter*, *packet loss*)[16]. (Botta, Donato, Dainotti, Avallone, & Pescapé, 2013)

Puede generar tráfico siguiendo modelos estocásticos, especificando la distribución que se desea escogiendo entre valores aleatorios para generar tráfico y posteriormente obtener un resumen estadístico. D-ITG es compatible y soporta los siguientes protocolos:

- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- SCTP (Stream Control Transmission Protocol)
- DCCP (Datagram Congestion Control Protocol)
- ICMP (Internet Control Message Protocol)

2.3.3.1 Arquitectura

Las características principales de D-ITG son proporcionadas por ITGSend y por ITGRecv. ITGSend es el componente responsable de generar tráfico hacia ITGRecv. Por su diseño multiproceso, ITGSend puede enviar de forma paralela múltiples flujos de tráfico hacia múltiples instancias ITGRecv, así mismo, ITGRecv puede recibir múltiples flujos de tráfico paralelamente desde múltiples instancias ITGSend. Cada flujo dispone de su propia señalización para poder diferenciarse. (Botta, et al., 2013)

[16] Conocidos por sus nombres en inglés, se refiere a: (rendimiento, retardo, variabilidad temporal, pérdida de paquetes)

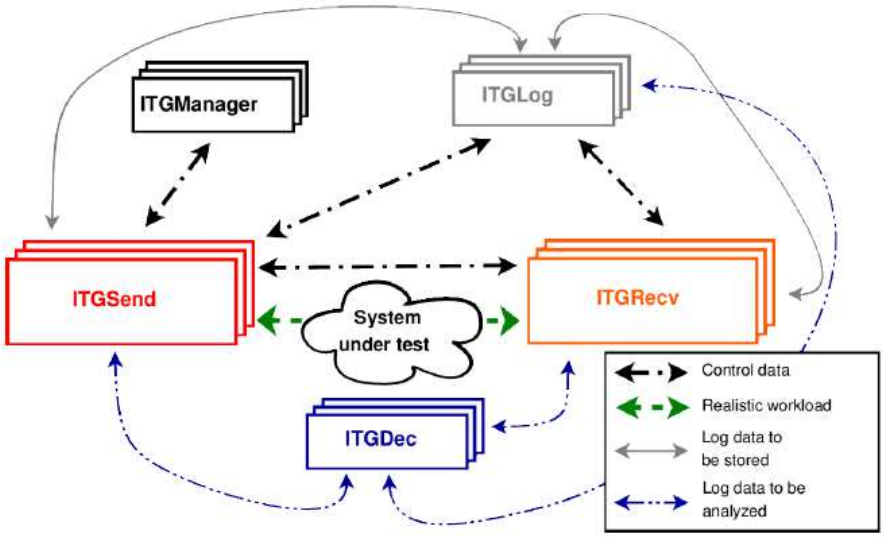

Figura 6 Arquitectura D-ITG. Por: (Alessio Botta, 2013)

ITGSend, ITGRecv pueden también producir archivos de registro que contiene información detallada acerca de cada paquete enviado y recibido. Estos archivos pueden guardarse localmente o enviarse a través de la red hacia el componente ITGLog, que sirve para recopilar todas las mediciones. El componente ITGDec está a cargo del análisis de los archivos de registro, a fin de extraer las métricas del rendimiento relacionado con los flujos del tráfico. (Botta, et al., 2013)

2.3.3.2 Características

D-ITG es capaz de generar múltiples flujos unidireccionales desde muchos remitentes hacia muchos receptores, cada uno de ellos posee las siguientes características:

- Nivel de Flujo Personalizado
  - Duración
  - Delay
  - Número total de paquetes
  - Número total de Kbyte
- Posee características de Capa 3
  - Protocolos IPv4, IPv6
  - Personalización de campos de cabecera

    *Dirección IP fuente y destino

*Interfaz fuente atada (para dispositivos con varias tarjetas de red)

*Valor inicial de TTL

*DS byte

- Recorrido NAT: FTP-como modo pasivo

- Posee características de Capa 4
  - Protocolos: TCP, UDP, ICMP, DCCP, SCTP
  - Personalización de campos de cabecera

*Número de puerto fuente y destino

- Posee características de Capa 7
  - Perfiles estocásticos predefinidos de IDT (Inter Departure Time) y PS (Packet Size)

*Telnet

*DNS

*Quake3

*CounterStrike (activo e inactivo)

*VoIP (G.711, G.729, G.723)

  - Contenido de carga útil: aleatorio o leído desde archivo
  - Procesos estocásticos soportados por IDT y PS

*Algunas distribuciones que están disponibles son: Uniforme, Constante, Exponencial, Pareto, Cauchy, Normal, Poisson, Gamma, Weibull

  - Métricas de QoS a nivel de paquetes

*Bitrate

*Packet rate

*Retraso (requiere sincronización de reloj)

*Tiempo de ida y vuelta

*Jitter

*Pérdidas de paquetes

De acuerdo a lo expuesto, D-ITG demuestra ser el más acertado para el caso que se plantea, la variedad de opciones de personalización permiten configurar diversidad de entornos de tráfico, una distribución en especial con la que se cuenta es la de Gauss

(Normal o de Campana), que resulta ser la que mejor representa el tráfico que ocurre en un Datacenter (Botta, Donato, Dainotti, Avallone, & Pescapé, 2013).

2.4 Métricas de Red

Para la validación del escenario propuesto se utilizarán métricas de red basadas en invariantes. De acuerdo con lo expresado en el paper: Determinación de Invariantes en Grandes Centros de Datos basados en Topología Fat-Tree; invariante se define como:

Ciertas propiedades que se deducen a partir de las variables y los parámetros que describen la red, y que se mantienen en las topologías de los Datacenter, para diversos escenarios y/o configuraciones (Ortiz et al., 2014).

Los invariantes, para el caso de una topología, son una expresión del funcionamiento de la red en ambientes reales, así por ejemplo la ruta más larga entre dos nodos, el número de enlaces óptimo o el mínimo número de enlaces que debe fallar antes de que un nodo pierda conectividad. Debido a esto, los invariantes se convierten en una referencia del funcionamiento real de un Datacenter, para los ambientes de simulación/emulación. (Ortiz et al., 2014)

Capítulo III:

Implementación de la Simulación

3.1 Descripción del Escenario

El escenario planteado consiste en una topología Fat-Tree con las siguientes características:

- Nivel jerárquico bien definido en tres capas: núcleo, distribución y acceso (*n*=3) y *(k=4).*
- Existe en cada capa *k* switches con *k* puertos de los cuales *k/2* se usaran en conexiones uplink y *k/2* para conexiones downlink.
- El máximo número de hosts es $k^3/4$
- Se analizará solamente el tráfico interno, y éste solo tendrá tres sentidos posibles: entre hosts de un mismo switch, entre hosts de diferentes switch pero que pertenecen al mismo *pod* y hosts que estén en diferentes *pods*.
- Tráfico unicast, para todos los pares de hosts que son seleccionados de forma aleatoria, pero basado en distribución de probabilidad uniforme.
- Los flujos de datos de tráfico son UDP y la distribución de probabilidad de los tiempos entre transmisiones debe ser exponencial, está distribución ha sido validada por (Benson, Aditya, & David A., 2010) para el caso de un Datacenter.

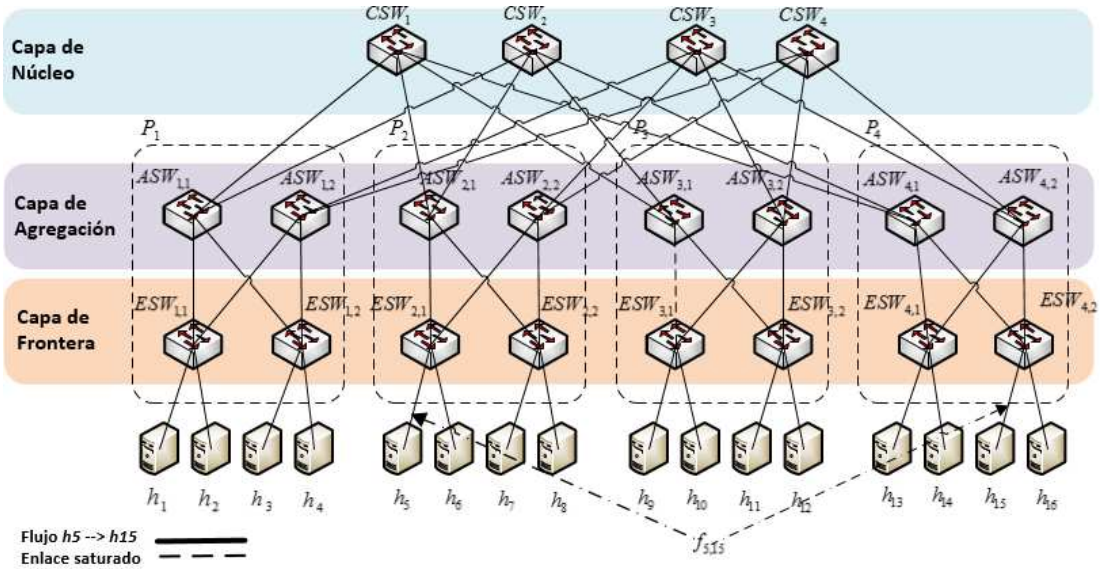

Figura 7 Topología Fat-Tree implementada. Por: (Ortiz et al., 2014)

3.2 Configuración de la Simulación

El escenario planteado fue planteado con ayuda de varias herramientas, como VirtualBox, donde se implementó una máquina virtual con Ubuntu 13.10 (64bits), sobre la cual se instaló el software Mininet, especializado en redes definidas por software. La gestión del enrutamiento se realizó a través del Controlador OpenDayLight, gracias a la destreza que brinda para la creación y selección de caminos redundantes. D-ITG, permitió generar múltiples flujos de tráfico de acuerdo a distribuciones estadísticas, similares a las que ocurren dentro de un Datacenter real. El analizador de paquetes Wireshark permitió la captura del tráfico para la recopilación de resultados. Los enlaces fueron configurados con capacidad máxima de 5 Mbps y los flujos con paquetes de tamaño constante e igual a 512 Bytes; esto con el fin de que la saturación de los enlaces llegue a ser posible en algún instante.

A continuación se detallan las versiones del software utilizado:

- VirtualBox 4.3.12
- Ubuntu Desktop 13.10 (64bit), 5GB-RAM
- OpenDayLight Hydrogen 1.0
- Mininet 2.1.0
- D-ITG-2.8.1-r1023
- Wireshark 1.10.2.

La implementación del escenario requiere que se instalen y ejecuten los programas indicados, para esto se deben seguir los pasos que se detallan en los Anexos desde el 1 hasta el 4.

El esquema de emulación será el siguiente:

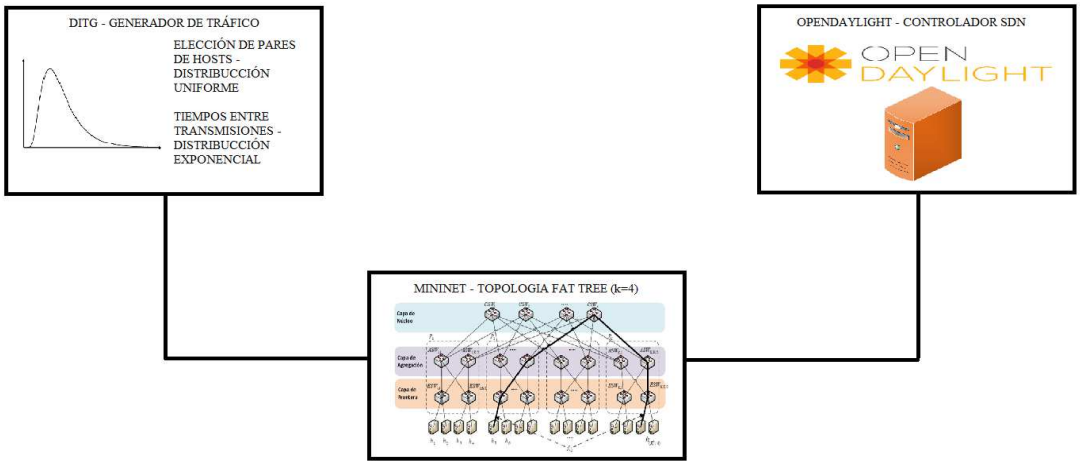

Figura 8 Esquema de simulación. Por: Los Autores

3.3 Validación del Escenario

En el artículo desarrollado por (Ortiz et al., 2014), se detallan las fórmulas que describen el análisis de métricas de red basadas en invariantes para una topología Fat-Tree (k=4).

3.3.1 Diámetro de la Red

Un invariante conocido es la longitud máxima del camino más corto entre todo par de h (hosts) de la red, o diámetro de la misma. La utilidad de este invariante radica en conocer el esfuerzo que tienen que hacer los paquetes de un flujo para viajar de extremo a extremo.

En una topología Fat-Tree, la mayor distancia se da cuando se comunican dos h que se encuentran en diferentes P (Pod), debido a que se necesita llegar hasta la capa más alta para dar paso al tráfico entre P. De la Fig. 9, para un $f_{5,15}$, el número de saltos requeridos en el tramo ascendente es igual a n-1 (no se considera al enlace entre h_5 y $ESW_{2,1}$ como salto), mientras que el número de saltos requeridos en el tramo descendente es igual a n-1.

Por lo tanto el tramo total para $f_{5,15}$ necesitará de un número total de saltos igual a 2n-1. Entonces D_{NET} queda definido de la siguiente forma:

$$D_{NET} = 2n - 1 \qquad (1)$$

3.3.2 Longitud de Ruta Promedio

Permite dimensionar el esfuerzo promedio que realizan todos los paquetes durante el desarrollo normal del tráfico de red. (p. 3)

Si se consideran todos los caminos más cortos entre cada uno de los hosts, y se los promedia sobre el número total de caminos, se obtiene la longitud de la ruta promedio L_{AV} de la topología, y cuya expresión matemática se define de la siguiente manera:

$$L_{AV} = \frac{\sum_{h_i=1}^{h_{MAX}-1}\left[\sum_{h_j=h_i+1}^{h_{MAX}} L(h_i, h_j)\right]}{\sum_{n=1}^{n-1} n} \qquad (2)$$

En donde L(hi,hj), es la longitud mínima o número de saltos mínimo entre dos hosts hi y hj. En esta expresión se garantiza que no existan pares de h repetidos. Para el Fat-Tree de la Figura 9, con k=4, el número de saltos mínimo solo puede tomar valores: 0 cuando los hosts pertenecen a un mismo switch, 3 cuando los hosts pertenecen a diferentes switches de un mismo P, y 5 cuando los hosts pertenecen a diferentes P. (p. 4)

Capítulo IV
Presentación de los Resultados

Considerando que los switches se presentan como equipos sin mayor inteligencia, ya que adquieren información e instrucciones acerca de la red sólo a través del controlador, por lo que no es posible medir esta invariante a través de una traza, donde normalmente se mostrarían los saltos de IP para llegar al destino, entonces mediante el software Wireshark se logró visualizar y/o capturar la información de las interfaces de red que se estén utilizando.

La prueba consistió en ejecutar un script [Anexo 8], que es capaz de generar dos diferentes clases de tráfico: un pequeño flujo de datos constante entre pares de hosts escogidos de forma aleatoria y a su vez generar un flujo constante de un peso mucho mayor que permite diferenciar que interfaces se encuentran participando para lograr que la información llegue al destino.

Se realizaron pruebas con los dos invariantes propuestos, para determinar su concordancia con lo analítico. Los resultados fueron los que se indican a continuación:

4.1 Validación mediante el Diámetro de la Red

El cálculo de D_{NET} experimental, se realizó tomando 100 pares de host de forma aleatoria, en cada par se ejecutó el script de generación de tráfico, mediante Wireshark se observaron las interfaces que estuvieron transmitiendo la mayor cantidad de paquetes durante la prueba y de esta manera se obtuvo el número de saltos (D_{NET}), siendo los resultados en promedio de 3 y 5, estos resultados están dentro del valor máximo del D_{NET} estimado de forma analítica. En la tabla 3 se exponen las interfaces que intervinieron en el recorrido de 10 pares tomados como ejemplo y la cantidad de saltos que se requieren para llegar al destino.

Tabla 3
Trazas parciales para DNET

CASO	TRAZA	D_{NET}
1	$h_1 \rightarrow ESW_{1,1} \rightarrow ASW_{1,2} \rightarrow CSW_3 \rightarrow ASW_{2,2} \rightarrow ESW_{2,1} \rightarrow h_5$	5
2	$h_2 \rightarrow ESW_{1,1} \rightarrow ASW_{1,2} \rightarrow CSW_3 \rightarrow ASW_{3,2} \rightarrow ESW_{3,1} \rightarrow h_{10}$	5
3	$h_4 \rightarrow ESW_{1,2} \rightarrow ASW_{1,1} \rightarrow CSW_2 \rightarrow ASW_{4,1} \rightarrow ESW_{4,1} \rightarrow h_{13}$	5
4	$h_8 \rightarrow ESW_{2,2} \rightarrow ASW_{2,1} \rightarrow CSW_2 \rightarrow ASW_{4,1} \rightarrow ESW_{4,2} \rightarrow h_{15}$	5
5	$h_2 \rightarrow ESW_{1,1} \rightarrow ASW_{1,1} \rightarrow ESW_{1,2} \rightarrow h_4$	3
6	$h_{11} \rightarrow ESW_{3,2} \rightarrow ASW_{3,1} \rightarrow CSW_1 \rightarrow ASW_{4,1} \rightarrow ESW_{4,1} \rightarrow h_{14}$	5
7	$h_{16} \rightarrow ESW_{4,2} \rightarrow ASW_{4,1} \rightarrow CSW_2 \rightarrow ASW_{2,1} \rightarrow ESW_{2,2} \rightarrow h_7$	5
8	$h_5 \rightarrow ESW_{2,1} \rightarrow ASW_{2,1} \rightarrow CSW_1 \rightarrow ASW_{3,1} \rightarrow ESW_{3,1} \rightarrow h_9$	5
9	$h_{10} \rightarrow ESW_{3,1} \rightarrow ASW_{3,1} \rightarrow ESW_{3,2} \rightarrow h_{11}$	3
10	$h_{13} \rightarrow ESW_{4,1} \rightarrow ASW_{4,1} \rightarrow ESW_{4,2} \rightarrow h_{16}$	3

Nota: Se detallan las interfaces que intervinieron en el recorrido de los 10 pares de hosts. Por: (Ortiz, et al., 2014)

La figura 9, resume de forma estadística, el resultado de las 100 iteraciones realizadas.

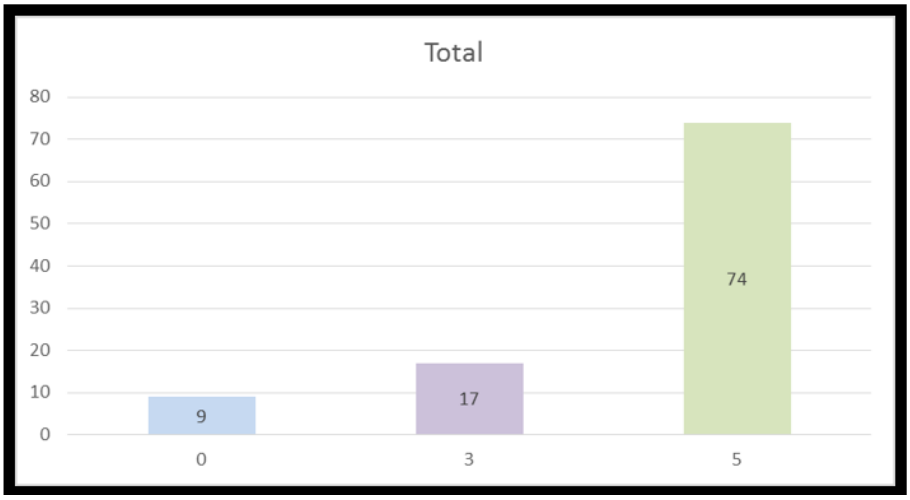

Figura 9 Estadística del máximo de número de saltos. Por: (Ortiz, et al., 2014)

Debido a que porcentualmente el mayor número de saltos es 5, se concluye que el D_{NET} experimental es igual a 5. Para el cálculo analítico del D_{NET}, se considera la ecuación (1), y por lo que se obtiene:

$$D_{NET} = 2(3)-1 = 5$$

Como se puede observar en la Tabla 4 los valores experimentales y analíticos coinciden, con lo cual la métrica basada en el invariante D_{NET} queda comprobada.

Tabla 4
Comparación DNET

D_{NET} Experimental	D_{NET} Analítico
5	5

Nota: Se comparan los resultados experimentales y analíticos de DNET. Por: Los autores.

4.2 Validación mediante Longitud de la Ruta Promedio

Para obtener los resultados se tomaron los mismos 100 valores obtenidos de la prueba experimental anterior, en este caso se realiza la suma total del número de saltos que utilizaron todos los pares de hosts, cuyo valor fue 421, esta cantidad se divide para el número total de muestras (100), lo cual da como resultado 4.21 que corresponde al L_{AV} experimental. Se infiere de acuerdo a este resultado que se requiere en promedio 4 saltos para que una pareja de hosts se comunique.

A partir de la ecuación (2), se determina el L_{AV} analítico, se deben calcular el número de saltos de cada par de hosts, sin tomar en cuenta las parejas repetidas y sumar los valores En este caso el resultado fue de 528 como se puede apreciar en la Tabla 5

Tabla 5
Número de saltos para cada par de hosts

HOSTS	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15	H16	
H1		0	3	3	5	5	5	5	5	5	5	5	5	5	5	5	66
H2			3	3	5	5	5	5	5	5	5	5	5	5	5	5	66
H3				0	5	5	5	5	5	5	5	5	5	5	5	5	60
H4					5	5	5	5	5	5	5	5	5	5	5	5	60
H5						0	3	3	5	5	5	5	5	5	5	5	46
H6							3	3	5	5	5	5	5	5	5	5	46
H7								0	5	5	5	5	5	5	5	5	40
H8									5	5	5	5	5	5	5	5	40
H9										0	3	3	5	5	5	5	26
H10											3	3	5	5	5	5	26
H11												0	5	5	5	5	20
H12													5	5	5	5	20
H13														0	3	3	6
H14															3	3	6
H15																0	0
H16																	0
SUMA																	528
NÚMERO DE PRUEBAS REALIZADAS																	120
PROMEDIO																	4,4

Nota: Se detallan los saltos necesarios para llegar a cualquier host de la red. Por: Los autores.

El total posteriormente se divide para el número total de pares escogidos o el número total de veces en que se realizó la prueba, obteniendo lo siguiente:

$$L_{AV} = 528/120 = 4.4$$

Esto implica que el promedio del L_{AV} analítico es 4 saltos para lograr la comunicación de un par de hosts cualquiera.

Se considera probada la métrica basada en el invariante L_{AV}, tomando en cuenta que los valores experimentales y analíticos fueron coincidentes tal como se detalla en la Tabla 6.

Tabla 6
Comparación LAV

L_{AV} Experimental	L_{AV} Analítico
$4.21 \approx 4$	$4.44 \approx 4$

Nota: Se comparan los resultados experimentales y analíticos de LAV. Por: Los autores Por: Los autores

Conclusiones

Las métricas de red han sido detalladas analíticamente, y luego comparadas con los resultados obtenidos mediante un emulador de tráfico en condiciones similares al de un Datacenter real, resultando ser ambos valores muy similares, esto confirma que el escenario bajo las condiciones planteadas es válido.

En el presente trabajo se ha propuesto el uso de métricas de red como método de verificación de modelos concebidos en ambientes de simulación, pues al ser estas métricas una expresión de una red real, permiten establecer la fidelidad del modelo desarrollado para topologías de Datacenter.

Este trabajo se ha limitado a utilizar como escenario un Datacenter con topología Fat-Tree. Sin embargo el mismo procedimiento de comprobación puede utilizarse para futuros trabajos que involucren el uso de diferentes topologías o controladores de Redes Definidas por Software.

La validación del escenario depende fuertemente de las métricas de red basadas en invariantes que se elijan, debido eso es fundamental que estas sean producto de un análisis profundo de la topología de red a usar.

Recomendaciones

Se recomienda apegarse a los manuales que se incluyen en los anexos de este documento, debido a que durante el desarrollo de este proyecto fueron analizadas varias herramientas con funciones similares pero con diferentes características, por lo tanto, lo propuesto es lo más adecuado para el escenario deseado.

Mientras se realiza el proceso de experimentación, se debe ejecutar el script de generación de tráfico varias veces, así que se recomienda antes de iniciar una nueva prueba ejecutar el script para matar los procesos.

En el caso de necesitar un escenario más grande, se recomienda utilizar computadores o servidores con mayor capacidad de procesamiento, memoria RAM y almacenamiento, para agilitar los tiempos de respuesta.

Cronograma

Tabla 7
Cronograma

ACTIVIDAD	DURACION DE LA ACTIVIDAD
Exponer las características principales de la topología más sencillas de implementar en casos de estudio y Data Centers: Fat-Tree.	2 SEMANAS
Utilizar el software Mininet como herramienta de simulación para entornos de redes definidas por software	8 SEMANAS
Emular a Fat-Tree con Mininet y generar trafico estadístico	8 SEMANAS
Validar el escenario planteado utilizando métricas de red bien conocidas	3 SEMANAS
Exponer el análisis acerca de la factibilidad de usar la herramienta Mininet como desarrollo y estudio de redes definidas por software	2 SEMANAS
Realizar la documentación de soporte	1 SEMANA

Nota: Se detalla la duración de cada actividad. Por: Los autores

Presupuesto

Tabla 8
Presupuesto

COTIZACIÓN	
RUBROS	$
ESTUDIO DEL PROYECTO	
Libros	$50
Seminarios y Cursos	$100
IMPLEMENTACION	
Instrumentos técnicos	$800
Materiales Varios	$30
VARIOS	
Movilización	$100
TOTAL	**$1,080**

Nota: Se detalla el costo aproximado de los rubros del proyecto. Por: Los autores

Bibliografía

Rojas A. & Pachón . (18 de Septiembre de 2013). *http://www.redclara.net/indico/evento/197.* Obtenido de Red CLara: http://bibliotecadigital.icesi.edu.co/biblioteca_digital/bitstream/10906/68434/8/articulo-redes-sdn.pdf.txt

Alea, A. (2013). *Manual de Linux.* Gijón, España.

Alessio Botta, W. d. (28 de October de 2013). D-ITG Manual. *D-ITG 2.8.1 Manual*. Napoli.

assessing-mininet. (2014). *GitHub*. Obtenido de https://github.com/sjas/assessing-mininet

Astudillo A., Dreier G., & Tarouco L. (2012). *Red CLARA.* Recuperado el 18 de Septiembre de 2012, de http://www.redclara.net/indico/evento/197

Belmar Mex Uc. (2008). Balanceo Distribuido del Encaminamiento para topologías Fat-tree sobre redes Infiniband.

Benson, T., Aditya, A., & David A., M. (Noviembre de 2010). Network Traffic Characteristics of Data Centers in the Wild.

Botta, A., Donato, W. d., Dainotti, A., Avallone, S., & Pescapé, A. (28 de Octubre de 2013). D-ITG 2.8.1 Manual. Italia: University of Napoli Federico II.

Centro de apoyo tecnológico a Emprendedores - Bilib. (2011). *Análisis de aplicación: VirtualBox.* Castilla-La Mancha: Creative Commons.

Garcés, H. (2000). *Investigación Científica.* Quito, Ecuador: Ediciones Abya-Yala.

GitHub. (2014). *GitHub*. Obtenido de https://gist.github.com/pichuang/9875468

Greenfield, L. (1996). *Guía de Linux para el usuario.* New Jersey, USA: Larry Greenfield - Grupo LuCAS.

Hernández, R., Fernández, C., & Baptista, P. (1991). *Metodología de la Investigación.* Naucalpan de Juárez, Estado de México: MCGRAW-HILL.

Lantz, B., Handigol, N., Heller, B., & Jeyakumar, V. (18 de Diciembre de 2014). *GitHub*. Obtenido de Introduction to Mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

Linux Foundation. (2014). *Linux Foundation*. Obtenido de Open Daylight: http://www.opendaylight.org/project/technical-overview

Liu, Y., Muppala, J. K., Veeraghavan, M., Lin, D., & Hamdi, M. (2013). Data Center Networks Topologies, Architectures and Fault-Tolerance Characteristics. New York - London: Springer.

Malhotra, N. K. (2008). *Investigación de mercados.* Naucalpan de Juárez, Estado de México: Pearson Educación.

MininetProject. (2015). Obtenido de https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

OpenDaylight. (22 de Diciembre de 2014). *OpenDaylight*. Obtenido de https://wiki.opendaylight.org/view/OpenDaylight_Controller

Oracle Corporation. (2004). *Oracle VM VirtualBox User Manual.* Redwood Shores, California, USA: Oracle Corporation.

Ortiz, J., Ampuño, A., Chávez, M., Londoño, J., & Novillo, F. (12 de Noviembre de 2014). *Determinación de Invariantes en Grandes Centros de Datos basados en Topología Fat-Tree.* Guayaquil.

Ramírez, A. (2004). *Metodología de la Investigación Científica.* Bogotá, Colombia: Pontificia Universidad Javeriana.

Ramírez, E. (s.f.). Manual Básico de Ubuntu. *Proyecto de Fortalecimiento de las Capacidades en Tecnologías de Información y Comunicación en Pequeñas y Medianas Empresas y Gobiernos Locales mediante el uso del Software Libre*. Costa Rica: Creative Commons.

Spera, C. (Marzo de 2013). Software Defined Network: el futuro de las arquitecturas de red. Obtenido de http://www.la.logicalis.com/globalassets/latin-america/logicalisnow/revista-20/lnow20-nota-42-45.pdf

Ubuntu Manual. (2010). *Primeros pasos con Ubuntu 12.04.* San Francisco, California, USA.: Creative Commons.

Wiki OpendayLight. (Abril de 2014). Obtenido de https://wiki.opendaylight.org/view

Xavier , L. C. (Septiembre de 2012). Representación gráfica del funcionamiento de redes de interconexión Fat-Tree.

Anexos

Anexo 1 Artículo científico aprobado para publicación en el Primer Congreso Internacional de Sociedades de la Información y Telecomunicaciones en el año 2014

Determinación de Invariantes en Grandes Centros de Datos Basados en Topología Fat-Tree

Ortiz J.*; Londono J.; Novillo F.***; Ampuno A.*; Chávez M.***

**Universidad Politécnica Salesiana, Facultad de Ingenierías, Guayaquil, Ecuador*
jortiz@ups.edu.ec; {aampuno; mchavezc}@est.ups.edu.ec
*** Universidad Pontificia Bolivariana, Facultad de Ingeniería en Tecnologías de la Información y la Comunicación, Medellín, Colombia*
jorge.londono@upb.edu.co
**** Escuela Superior Politécnica del Litoral, CIDIS-Facultad de Ingeniería en Electricidad y Computación, Guayaquil, Ecuador*
fnovillo@espol.edu.ec

Resumen: Durante los últimos años ha existido un fuerte incremento en el acceso a internet, causando que los centros de datos (*DC*) deban adaptar dinámicamente su infraestructura de red de cara a enfrentar posibles problemas de congestión, la cual no siempre se da de forma oportuna. Ante esto, nuevas topologías de red se han propuesto en los últimos años, como una forma de brindar mejores condiciones para el manejo de tráfico interno, sin embargo es común que para el estudio de estas mejoras, se necesite recrear el comportamiento de un verdadero *DC* en modelos de simulación/emulación. Por lo tanto se vuelve esencial validar dichos modelos, de cara a obtener resultados coherentes con la realidad. Esta validación es posible por medio de la identificación de ciertas propiedades que se deducen a partir de las variables y los parámetros que describen la red, y que se mantienen en las topologías de los *DC* para diversos escenarios y/o configuraciones. Estas propiedades, conocidas como invariantes, son una expresión del funcionamiento de la red en ambientes reales, como por ejemplo la ruta más larga entre dos nodos o el número de enlaces mínimo que deben fallar antes de una pérdida de conectividad en alguno de los nodos de la red. En el presente trabajo se realiza la identificación, formulación y comprobación de dos invariantes para la topología Fat-Tree, utilizando como software emulador a mininet. Las conclusiones muestran resultados concordantes entre lo analítico y lo práctico.

Palabras clave: Invariantes de red, topologías, Fat-tree, simulación, emulación.

Abstract: In recent years there has been a sharp increase in access to internet, causing data centers (*DC*) should dynamically adapt its network infrastructure to face possible problems of congestion, which is not always given in a timely manner. Given this, new network topologies have been proposed in recent years as a way to provide better conditions for handling internal traffic, however it is common for the study of these improvements the need to recreate the behavior of a real *DC* in models of simulation/emulation. Therefore it becomes essential to validate these models, in order to obtain consistent results with reality. Such validation is possible through the identification of certain properties which are derived from the variables and parameters that describe the network and are maintained in the *DC* topologies for different scenarios and/or configurations. These properties, known as invariant, are an expression of the operation of the network in real environments, such as the longest path between two nodes or the minimum number of links that must fail before a loss of connectivity on one of the nodes of the network. In this paper, the identification, formulation and testing of two invariants for the

Fat-Tree topology is performed, using MiniNet as the software emulator. The conclusions show good agreement between the analytical and the practical.

Keywords: Network invariants, topologies, Fat-tree, simulation, emulation.

1. INTRODUCCION

Durante los últimos años ha existido un fuerte incremento en el acceso a internet provocado en gran parte por la extensa variedad de servicios que se ofrecen [8]. Este hecho ha causado que los *DC* se vean obligados a adaptar dinámicamente su infraestructura de red para soportar el incremento de tráfico generado; de otra manera se podrían provocar problemas de congestión, bajo rendimiento de las aplicaciones, alta tasa de errores y pérdida de paquetes, entre otros [20] [11] [19] [13]. Tal adaptación no siempre se da de forma oportuna y depende en muchos casos de nuevas inversiones en equipos de comunicación y/o enlaces, lo que supone también un incremento en la complejidad de la red, desde el punto de vista de gestión y configuración.

De cara a resolver estos problemas, en los últimos años nuevas topologías de red para los *DC* se han presentado [3], de manera que se brinden mejores condiciones para el manejo del tráfico interno, mediante la aplicación de mecanismos que incluyen la posibilidad de enrutamiento a través de rutas paralelas, acortamiento de distancias entre nodos y el aumento de tolerancia ante fallos, entre otros. Por ejemplo VL2 [9] evita el tráfico broadcast ARP y DHCP mediante el uso de servidores de directorio para la resolución de direcciones. Con PortLand [16] se propone un mecanismo en enrutamiento en capa 2 empleando las propiedades de la topología. Bcube [10] soporta varias aplicaciones que requieran de ancho de banda intensivo mediante múltiples caminos paralelos cortos entre cada par de hosts.

Sin embargo previo a despliegues de infraestructura de DC, es importante validar dichos escenarios en un ambiente lo más real posible. En este contexto, recrear el comportamiento de los *DC*, mediante modelos de simulación/emulación [7], permite predecir algunos aspectos cruciales de la operatividad de la red. Debido a esto, se vuelve esencial validar dichos modelos con el fin de que exista coherencia entre los resultados que se obtienen de la simulación/emulación y los resultados que se obtendrían de un *DC* real. Una técnica que permite validar estos modelos se basa en la determinación de invariantes, la cual consiste en identificar ciertas propiedades de la red, que se deducen a partir de las variables y/o parámetros que la describen, y que se mantienen en las topologías de los *DC*, para diversos escenarios y/o configuraciones.

Un invariante, así por ejemplo, es la ruta más larga entre dos nodos, el número de enlaces óptimo o el mínimo número de enlaces que deben fallar antes de que un nodo pierda conectividad. Debido a esto, los invariantes se convierten en una muy buena referencia del funcionamiento real de un *DC*, para los ambientes de simulación/emulación.

Una metodología para la obtención de invariantes implica hacer una revisión analítica de la topología, considerando ambientes con y sin tráfico, con pocos y muchos hosts y dispositivos de red. Finalmente considerar la interacción con los protocolos utilizados en la red, por ejemplo para acceso al medio, enrutamiento y transporte. De esta revisión se obtienen los invariantes candidatos.

En la comunidad científica se han realizado algunos trabajos al respecto, así por ejemplo en [14] se proponen diez invariantes para servidores web de internet, los cuales permiten caracterizar el tráfico HTTP y facilitar el estudio de mejoras en el rendimiento del servidor. La problemática de realizar simulaciones en la red Internet se expone ampliamente en [20] formulando dentro de sus soluciones, la búsqueda de invariantes, y en la que se proponen siete alternativas: una de ellas relacionada a la topología. El tráfico generado por juegos de red en Internet es analizado en [1], determinando tres invariantes para el tiempo entre paquetes (IPT). Una metodología para el diseño de arquitecturas de red tomando en cuenta a los invariantes se propone en [4]; en este caso se los considera puntos fijos que limitan la evolución de las arquitecturas

Por lo tanto, en el presente trabajo se propone identificar, formular y comprobar, dos

invariantes de red para grandes centros de datos, considerando una de las topologías más utilizadas en el análisis del comportamiento del tráfico en *DC*, como lo es Fat-Tree. En particular esta topología será emulada utilizando software ampliamente aceptado en la comunidad científica, como lo es mininet [6], y mediante el uso de una herramienta de generación de tráfico, se modelará un escenario que permite recrear casos reales de tráfico en un *DC*. Esto permitirá comprobar el cumplimiento de los invariantes formulados, sobre un escenario emulado. El resto del documento está organizado como se explica a continuación. En la sección II se describirá el escenario de la topología sobre la cual se va a trabajar. En la sección III, dos invariantes serán propuestos de forma analítica. En la sección IV los invariantes serán comprobados experimentalmente, y comparados con los obtenidos analíticamente. Finalmente las conclusiones del artículo son presentadas.

2. MODELADO DEL SISTEMA

2.1 Descripción del escenario

El escenario bajo estudio considera un *DC* con topología Fat-Tree de Al-Fares [15], una instancia de la topología Clos [21]. Esto debido a que en los últimos años, se ha referenciado su uso en varias propuestas de la comunidad científica.

Fat-Tree está compuesta fundamentalmente por switches de iguales características y que se encuentran organizados jerárquicamente en *n* capas denominadas de núcleo, de agregación y de frontera . Adicionalmente entre estas dos últimas se forman agrupaciones entre switches denominados Pods (*P*). Al interior de los P, los switches compuestos por *k* puertos, utilizan *k*/2 puertos para hacer uplink y *k*/ 2 puertos para hacer downlink. En la Fig. 1 se puede apreciar una topología Fat-Tree con *n*=3. La capa de núcleo está compuesta por un conjunto de *k* switches (*CSW*) con *k* puertos activos que permiten el tráfico hacia y desde redes externas con el resto del *DC*, así como tráfico interno entre *P*. A partir de aquí los hosts (*h*) se conectarán a los P_K, utilizando los *k*/2 puertos disponibles de los switches de la capa de frontera (*ESW*). El número máximo de hosts h_{MAX} es igual a $K_3/4$.

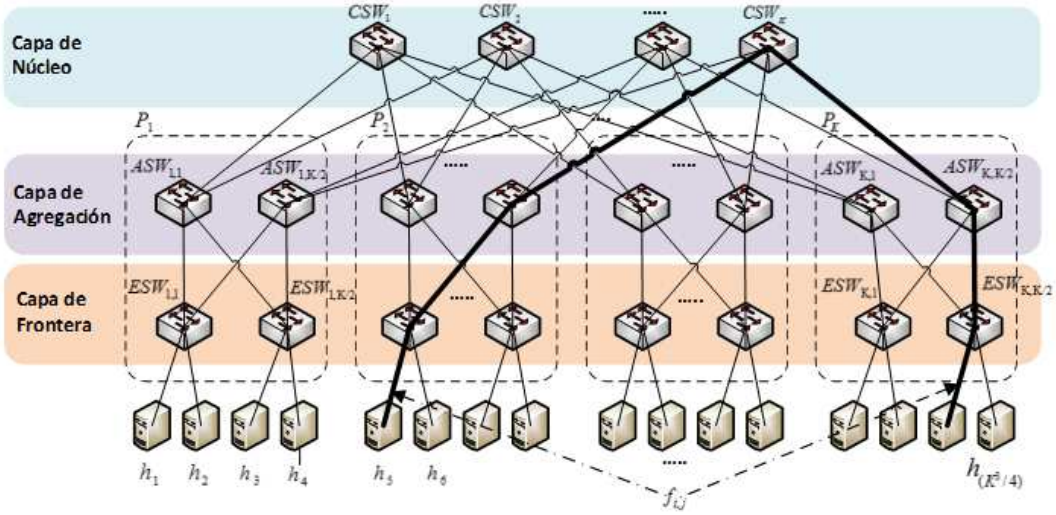

Figura 1 Topología Fat-Tree

3. FORMULACIÓN DE LOS INVARIANTES

Para formular adecuadamente los invariantes primero es necesario conocer cómo funciona la topología y luego identificar características de la red que puedan ser debidamente cuantificadas y representadas en expresiones matemáticas, ya que en ambientes experimentales, los invariantes validarán la simulación/emulación. Los siguiente invariantes se proponen para un *DC* con Fat-Tree, sin embargo se fundamentan en

conceptos que pueden ser aplicables a otras topologías. Estos invariantes se relacionan con algunos de los parámetros característicos de las topologías [12], y son los siguientes:

3.1 Diámetro de la red

Un invariante conocido es la longitud máxima del camino más corto entre todo par de h de la red, o diámetro de la misma. La utilidad de este invariante radica en conocer el esfuerzo que tienen que hacer los paquetes de un flujo para viajar de extremo a extremo.

En una topología Fat-Tree, la mayor distancia se da cuando se comunican dos h que se encuentran en diferentes *P*, debido a que se necesita llegar hasta la capa más alta para dar paso al tráfico entre *P*. De la Fig. 1, para un $f_{5,15}$, el número de saltos requeridos en el tramo ascendente es igual a *n*-1 (no se considera al enlace entre h_5 y $ESW_{2,1}$ como salto), mientras que el número de saltos requeridos en el tramo descendente es igual a *n*-1. Por lo tanto el tramo total para $f_{5,15}$ necesitará de un número total de saltos igual a 2*n*-1. Entonces D_{NET} queda definido de la siguiente forma:

$$D_{NET} = 2n-1 \qquad (1)$$

Por otro lado, este invariante no considera que ante un mayor flujo de información, la probabilidad de saturación de los enlaces también aumentará, obligando al mecanismo gestor del enrutamiento de la red, seleccionar rutas alternas que faciliten la libre circulación de los datos. Muchas veces el tomar estas rutas alternas, implica enviar los paquetes por un tramo más largo de extremo a extremo, por lo que el número de saltos para un fi,j también aumenta. En la Fig. 2 se puede apreciar un caso extremo para f5,15 en donde todos los posibles caminos redundantes se han saturado, dejando como única alternativa para el encaminamiento de los paquetes, a la señalada en la Fig. 2.

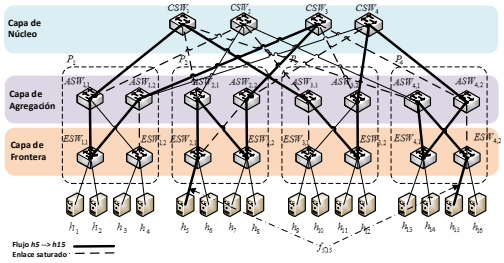

Figura 2 Topología Fat-Tree con enlaces saturados

Es en este caso que se puede determinar un D'NET que sí considere la saturación de los enlaces. En la Fig. 3 se muestra otra perspectiva de la red, con los ascensos y descensos entre capas que realiza f5,15 en su recorrido de extremo a extremo. Los tramos ascendentes o descendentes que atraviesan una capa, tienen un número de saltos igual a n-2, mientras que los tramos ascendentes o descendentes que atraviesan dos capas, tienen un número de saltos igual a n-1. A esto hay que adicionar un salto correspondiente a la llegada al h15.

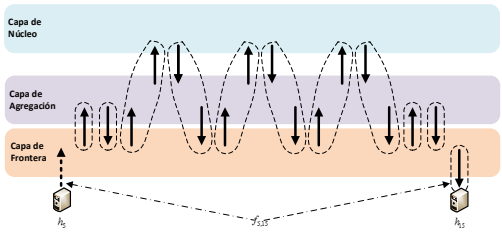

Figura 3. Resumen de saltos entre capas para f5,15.

Por lo tanto el invariante D'NET se define de la siguiente forma:

$$D'_{NET} = 2(n-2)+6(n-1)+2(n-2)+1$$
$$D'_{NET} = 10n-13 \qquad (2)$$

3.2 Longitud de ruta promedio

Otro invariante de utilidad es la longitud de ruta promedio LAV, el cual permite dimensionar el esfuerzo promedio que realizan todos los paquetes durante el desarrollo normal del tráfico de red.

Si se consideran todos los caminos más cortos entre cada uno de los hosts, y se los promedia sobre el número total de caminos, se obtiene la longitud de la ruta promedio LAV de la topología, y cuya expresión matemática se define de la siguiente manera:

$$L_{AV} = \frac{\sum_{h_i=1}^{h_{MAX}-1}\left[\sum_{h_j=h_i+1}^{h_{MAX}} L(h_i,h_j)\right]}{\sum_{n=1}^{n-1} n} \qquad (3)$$

Donde *L(hi,hj)*, es la longitud mínima o número de saltos mínimo entre dos hosts h_i y h_j. En esta expresión se garantiza que no existan pares de *h* repetidos. Para el Fat-Tree de la Fig. 1 con *k*=4, el número de saltos mínimo solo puede tomar tres valores: 0 cuando los hosts pertenecen a un mismo switch, 3 cuando los hosts pertenecen a diferentes switches de un mismo *P*, y 5 cuando los hosts pertenecen a diferentes *P*.

4. COMPROBACIÓN DE LOS INVARIANTES

En esta sección se describe la forma en como el escenario fue emulado, y luego como se contrastan los resultados experimentales con los analíticos.

4.1 Configuración de la emulación

Para recrear el escenario propuesto se utilizaron varias herramientas; así para emular la topología Fat-Tree se escogió al software mininet [6], pues con la llegada de las redes definidas por software (SDN), es uno de los programas comúnmente utilizado para emular topologías de *DC* en los últimos años [17]. Para gestionar el enrutamiento, se utiliza el controlador SDN OpenDaylight [18] por la facilidad que brinda en la creación y selección de caminos redundantes por sobre otros controladores. El generador de tráfico D-ITG [2], permitió emular los múltiples flujos de información característica de un *DC* y mediante el analizador de paquetes Wireshark [22] se capturó el tráfico para la recopilación de resultados. Cada uno de los enlaces fueron ajustados a 5 Mbps y los flujos con paquetes de tamaño constante e igual a 512 Bytes; esto con el fin de que la saturación de los enlaces llegue a ser posible en algún instante.

Para la determinación del DNET experimental, se procedió a generar un flujo con un mayor tamaño de paquete, para facilitar la identificación entre los demás flujos del *DC*. Para obtener el L_{AV} se generaron flujos entre cada uno de los pares de h hasta completar un número de muestras representativo.

4.2 Análisis de resultados

Luego de que estos invariantes fueron calculados de forma experimental, se obtuvieron los resultados abajo descritos.

Diámetro de la red

Para calcular D_{NET} experimental, se tomaron 100 pares de host al azar y de acuerdo a la traza generada por el $f_{i,j}$, se obtuvieron como máximo número de saltos, los valores de 0, 3 y 5. En la Fig. 4 y en la Fig. 5 se puede apreciar la traza generada por los 10 primeros valores y la estadística del número máximo de saltos para todas las muestras, respectivamente.

CASO	TRAZA	DNET
1	$h_1 \rightarrow ESW_{1,1} \rightarrow ASW_{1,2} \rightarrow CSW_3 \rightarrow ASW_{2,2} \rightarrow ESW_{2,1} \rightarrow h_5$	5
2	$h_2 \rightarrow ESW_{1,1} \rightarrow ASW_{1,2} \rightarrow CSW_3 \rightarrow ASW_{3,2} \rightarrow ESW_{3,1} \rightarrow h_{10}$	5
3	$h_4 \rightarrow ESW_{1,2} \rightarrow ASW_{1,1} \rightarrow CSW_2 \rightarrow ASW_{4,1} \rightarrow ESW_{4,1} \rightarrow h_{13}$	5
4	$h_8 \rightarrow ESW_{2,2} \rightarrow ASW_{2,1} \rightarrow CSW_2 \rightarrow ASW_{4,1} \rightarrow ESW_{4,2} \rightarrow h_{15}$	5
5	$h_2 \rightarrow ESW_{1,1} \rightarrow ASW_{1,1} \rightarrow ESW_{1,2} \rightarrow h_4$	3
6	$h_{11} \rightarrow ESW_{3,2} \rightarrow ASW_{3,1} \rightarrow CSW_1 \rightarrow ASW_{4,1} \rightarrow ESW_{4,1} \rightarrow h_{14}$	5
7	$h_{16} \rightarrow ESW_{4,2} \rightarrow ASW_{4,1} \rightarrow CSW_2 \rightarrow ASW_{2,1} \rightarrow ESW_{2,2} \rightarrow h_7$	5
8	$h_5 \rightarrow ESW_{2,1} \rightarrow ASW_{2,1} \rightarrow CSW_1 \rightarrow ASW_{3,1} \rightarrow ESW_{3,1} \rightarrow h_9$	5
9	$h_{10} \rightarrow ESW_{3,1} \rightarrow ASW_{3,1} \rightarrow ESW_{3,2} \rightarrow h_{11}$	3
10	$h_{13} \rightarrow ESW_{4,1} \rightarrow ASW_{4,1} \rightarrow ESW_{4,2} \rightarrow h_{16}$	3

Figura 4 Trazas parciales para DNET

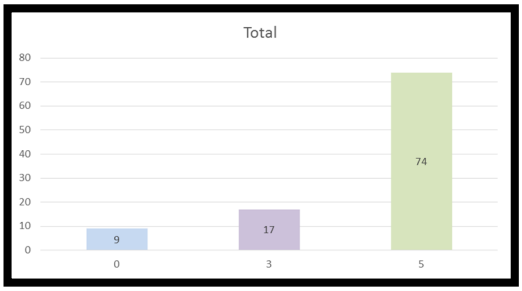

Figura 5 Estadística del máximo número de saltos

Siendo 5 el valor más alto dentro del número de saltos generados en la topología, el D_{NET} experimental es igual a 5.

Por otro lado para el cálculo del D_{NET} analítico, si se considera la ecuación (1) en la topología emulada con *n*=3, el valor para D_{NET} es igual a:

$$D_{NET} = 2(3) - 1 = 5 \quad (4)$$

Con lo cual el invariante D_{NET} queda comprobado. Cabe indicar que a pesar que de

que se ha manejado un gran volumen de tráfico, este no ha sido suficiente para saturar los enlaces y llegar a valores mayores y cercanos a un D'_{NET} experimental.

Longitud de ruta promedio

El cálculo del L_{AV} experimental se realizó sobre las 100 muestras anteriormente tomadas. La suma total del número de saltos que utilizaron todos los pares de hosts es igual a 421, cantidad que al dividirla para el número total de muestras (100), resulta en un L_{AV} experimental de 4,21. La interpretación de esto es que en promedio se requieren de 4 saltos para llegar a comunicar un par de hosts cualquiera de la topología.

En cambio, para determinar el L_{AV} analítico en la topología emulada, la ecuación (3) indica que primero se deben sumar el número de saltos de cada uno de los pares de hosts (no repetidos) que componen la topología. Esta suma es igual a 528 y se puede apreciar en la Fig. 6.

HOSTS	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15	H16	∑
H1		0	3	3	5	5	5	5	5	5	5	5	5	5	5	5	66
H2			3	3	5	5	5	5	5	5	5	5	5	5	5	5	66
H3				0	5	5	5	5	5	5	5	5	5	5	5	5	60
H4					5	5	5	5	5	5	5	5	5	5	5	5	60
H5						0	3	3	5	5	5	5	5	5	5	5	46
H6							3	3	5	5	5	5	5	5	5	5	46
H7								0	5	5	5	5	5	5	5	5	40
H8									5	5	5	5	5	5	5	5	40
H9										0	3	3	5	5	5	5	26
H10											3	3	5	5	5	5	26
H11												0	5	5	5	5	20
H12													5	5	5	5	20
H13														0	3	3	6
H14															3	3	6
H15																0	0
H16																	0
																	528

Figura 6. Número de saltos por cada par de hosts.

Luego este resultado debe ser divido para el número total de pares considerados en este cálculo, el cual es igual a 120. Por lo tanto el valor analítico para L_{AV} es igual a:

$$L_{AV} = 528/120 = 4.4 \quad (5)$$

Lo que implica, al igual que en el valor experimental, de que en promedio también se requiere de 4 saltos para que un par de hosts cualquiera se comuniquen.

5. CONCLUSIONES

En el presente trabajo se ha propuesto el uso de invariantes como método de verificación de modelos concebidos en ambientes simulación/emulación, pues al ser los invariantes una expresión de una red real, permiten establecer la fidelidad del modelo desarrollado para topologías de *DC*.

Se han propuesto dos invariantes, los cuales han sido demostrados analíticamente, y luego comparados mediante un emulador en condiciones de red similares al de un *DC* real, dando como resultado respuestas con un buen grado de aproximación.

Este trabajo se ha limitado a utilizar como escenario de un *DC*, la topología Fat-Tree, sin embargo se puede realizar las mismas verificaciones sobre otras topologías igualmente relevantes, como Bcube, VL2, Dcell, Portland, entre otras. Así mismo se pueden manejar otras distribuciones de probabilidad para la generación del tráfico, así como el uso de tráfico multicast o broadcast, propio de determinados servicios de red.

Es importante como un requisito previo para la determinación de invariantes, el conocimiento profundo de la topología a estudiar, pues de lo contrario los invariantes podrían deducirse incorrectamente.

REFERENCIAS

[1] Alberto Dainotti, Alessio Botta, Antonio Pescapé, and Giorgio Ventre. 2006. Searching for invariants in network games traffic. In Proceedings of the 2006 ACM CoNEXT conference (CoNEXT '06). ACM, New York, NY, USA., Article 43 , 2 pages, 1989.

[2] Avallone, S., Guadagno, S., Emma, D., Pescapè, A., & Ventre, G. (2004, September). D-ITG distributed internet traffic generator. In Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the (pp. 316-317). IEEE.

[3] Bari, M.F.; Boutaba, R.; Esteves, R.; Granville, L.Z.; Podlesny, M.; Rabbani, M.G.; Qi Zhang; Zhani, M.F., "Data Center Network Virtualization: A Survey," Communications Surveys & Tutorials, IEEE , vol.15, no.2, pp.909,928, Second Quarter 2013

[4] Bengt Ahlgren, Marcus Brunner, Lars Eggert, Robert Hancock, and Stefan Schmid. 2004. Invariants: a new design methodology for network architectures. In Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture (FDNA '04). ACM, New York, NY, USA, 65-70.

[5] Benson, T., Akella, A., & Maltz, D. A. (2010, November). Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (pp. 267-280). ACM.

[6] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). ACM, New York, NY, USA, , Article 19 , 6 pages.

[7] Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., ... & Yu, H. (2000). Advances in network simulation. *Computer*, *33*(5), 59-67.

[8] Chen, Z., Peng, L., Zhao, S., Zhang, L., & Jing, S. (2014). Feature Selection Toward Optimizing Internet Traffic Behavior Identification. In *Algorithms and Architectures for Parallel Processing* (pp. 631-644). Springer International Publishing.

[9] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., ... & Sengupta, S. (2009, August). VL2: a scalable and flexible data center network. In ACM SIGCOMM Computer Communication Review (Vol. 39, No. 4, pp. 51-62). ACM.

[10] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., ... & Lu, S. (2009). BCube: a high performance, server-centric network architecture for modular data centers. ACM SIGCOMM Computer Communication Review, 39(4), 63-74.

[11] Kurup, P. M., & Preethi, J. (2014). Packet Reordering To Improve Data Center Network Using Near Optimal Traffic Engineering.

[12] Lee, David S., and Jeffrey L. Kalb. "Network Topology Analysis."

[13] Luan, G. (2014). Buffer Stopping Time Analysis in Data Center Networks.

[14] Martin F. Arlitt and Carey L. Williamson. 1996. Web server workload characterization: the search for invariants. In Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '96), Blaine D. Gaither (Ed.). ACM, New York, NY, USA, 126-137

[15] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In Proceedings of the ACM SIGCOMM 2008 conference on Data communication (SIGCOMM '08). ACM, New York, NY, USA, 63-74.

[16] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., ... & Vahdat, A. (2009, August). Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review* (Vol. 39, No. 4, pp. 39-50). ACM.

[17] Nunes, B., Mendonca, M., Nguyen, X., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks.

[18] Orebaugh, A., Ramirez, G., & Beale, J. (2006). Wireshark & Ethereal network protocol analyzer toolkit. Syngress.

[19] Park, H. W., Yeo, I. Y., Lee, J. R., & Jang, H. (2014). Study on network architecture of big data center for the efficient control of huge data traffic. *Computer Science and Information Systems*, (00), 67-67.

[20] Sally Floyd and Vern Paxson. 2001. Difficulties in simulating the internet. IEEE/ACM Trans. Netw. 9, 4 (August 2001), 392-403.

[21] W. Dally and B. Towles, Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., 2004.

[22] WRIGHT, C., MESTERY, K., SHAIKH, A., & BAUCKE, S. OpenDaylight: An Open Source SDN for Your OpenStack Cloud. OpenStack Summit Presentations

Anexo 2: Instalación de generador de topologías Mininet

Opción 1: Instalación de Mininet VM (Virtual Machine).

Se puede descargar una imagen pre-construida de Mininet/Ubuntu. Incluye, Mininet, OpenFlow y herramientas pre-instaladas que incorpora ajustes para configuración del kernel que haga posible la creación de redes grandes. Es la forma más recomendada y fácil:

Se descarga la imagen virtual

https://github.com/mininet/mininet/wiki/Mininet-VM-Images

Se agrega al software virtual box creando una máquina virtual nueva.

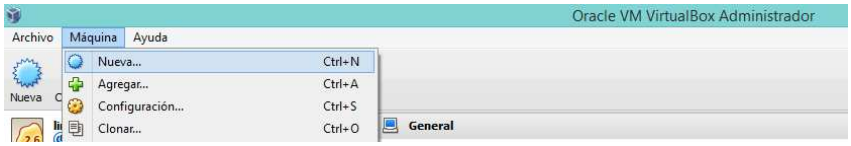

Figura 10 Crear nueva VM VirtualBox. Por: Los autores

Detallar el Nombre y el Sistema Operativo al que pertenece, en este caso Ubuntu.

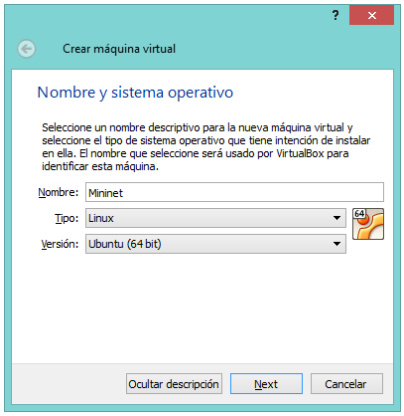

Figura 11 Selección de Sistema Operativo en VirtualBox. Por: Los autores

- Colocar el espacio de memoria RAM que utilizará la VM, se recomienda a partir de 2 Megas.

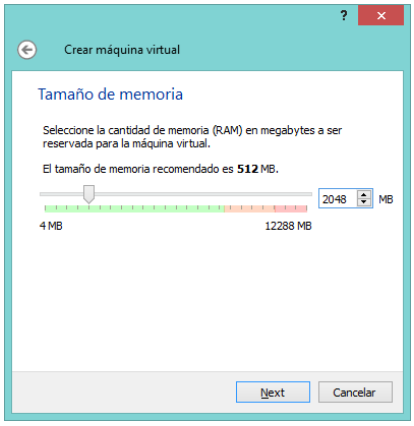

Figura 12 Configuración memoria RAM para el nuevo SO. Por: Los autores

Seleccionar la opción de disco virtual existente.

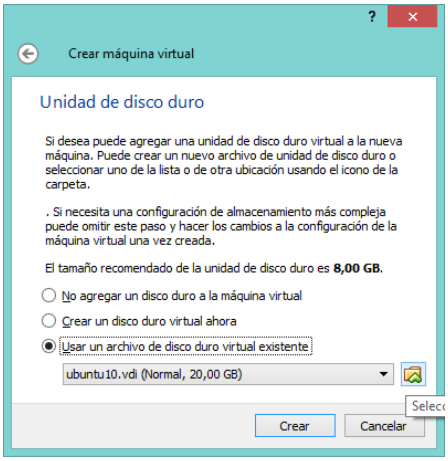

Figura 13 Elección de disco duro virtual para nueva VM. Por: Los autores

Escoger la imagen de Mininet que se ha descargado previamente y luego crear.

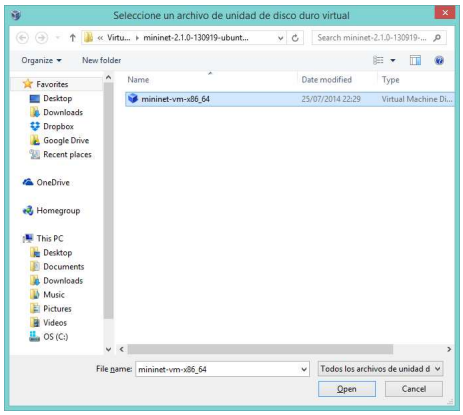

Figura 14 Elección de imagen descargada de Mininet. Por: Los autores

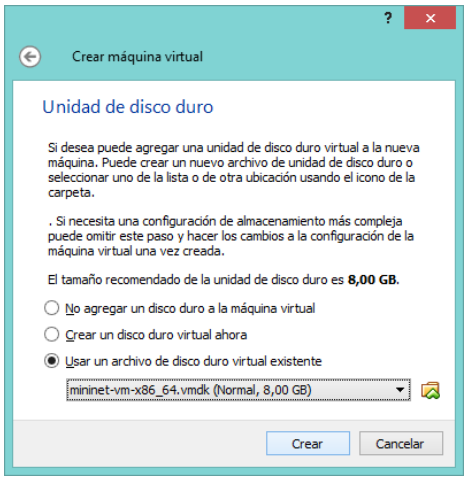

Figura 15 Selección de imagen para crear VM. Por: Los autores

- Iniciar la Máquina Virtual. Por defecto: login→Mininet, password→Mininet

Figura 16 Ejecución de Mininet VM. Por: Los autores

Opción 2: Instalación de Mininet nativa desde la fuente

Se requiere trabajar sobre SO Linux, recomendado Ubuntu en versiones recientes.

- En esta aplicación desde consola Ubuntu 13.10

  sudo apt-get install Mininet

- Instalación nativa desde la Fuente

  sudo git clone git://github.com/mininet/mininet.git

  El comando ***git*** verifica que la versión a descargar sea la última, si se desea obtener la versión última específica lanzada, usar:

  sudo git clone git://github.com/mininet/Mininet

  sudo git checkout -b 2.1.0 2.1.0

- Dirigirse al directorio */home/user/Mininet/util/*

  install.sh

  Asegurarse que para ejecutar este comando se encuentre en modo privilegiado: *root*

  Los paquetes instalados crean y modifican directorios importantes por lo cual se debe observar primero el script install.sh para verificar que se esté de acuerdo.

- Finalmente verificar a través del siguiente comando que la instalación haya sido exitosa.

  sudo mn --test pingall

Anexo 3: Instalación de controlador Opendaylight

Requerimientos de hardware y software

- Linux (Ubuntu, RHEL, Fedora o cualquier otra distribución de Linux que soporte Java)
- JVM 1.7+ (JAVA_HOME debe ajustarse según la implementación)
- El Controlador está construido en GUI. El GUI está implementado como una aplicación.

Instalación desde consola Ubuntu 13.10

- Se requiere ciertos componentes para compilar y descargar el Controlador. Asegurarse de encontrarse en modo privilegiado ***root*** para efectuar estos comandos

  apt-get update

- Luego, instalar los componentes que se requieren, entre estos JVM:

  apt-get install git

  apt-get install openjdk-7-jre

  apt-get install openjdk-7-jdk

  apt-get install maven

- Descargar el controlador

  http://git.opendaylight.org/gerrit/p/controller.git

- Compilar y construir el código del Controlador
- Entrar al directorio:

  cd controller/OpenDayLight/distribution/OpenDayLight/

- Compilar el Controlador

  mvn clean install

  Hasta este punto quizá se puede presentar un error de instalación, que tenga que ver con el espacio de memoria con el que se dispone, ya que estamos hablando de una VM. Para esto, esta línea de comando nos puede ayudar.

  export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m

- Establecer la variable JAVA_HOME, para que el Controlador pueda encontrar los components JDK que necesita.

  export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386/

  Este comando establece el requerimiento únicamente en la sesión activa, es decir que éste debe ejecutarse cada vez que se inicie la máquina virtual, pero en Linux se dispone de la ventaja de que los archivos de sistema pueden ser editados, para esto debemos primero tener instalado el paquete ***gedit:***

  sudo apt-get install gedit

- Luego de instalar el paquete, dirigirse al directorio */home/user* y colocar:

  sudo gedit .bashrc

- En el documento abierto colocar la siguiente línea a final:

  export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386/

Anexo 4: Instalación de generador de tráfico DITG

Independientemente del Sistema operativo sobre el cual se esté trabajando, se deben seguir los siguientes pasos:

- Descargar y descomprimir el archivo "D-ITG-2.8.1-r1023-src.zip". Página recomendada:

  http://traffic.comics.unina.it/software/ITG/download.php

Si el programa se ha descargado en una PC con una distribución de Linux se deberá realizar lo siguiente:

- Dirigirse al directorio D-ITG-2.8.1-r1023/src

cd home/usr/D-ITG-2.8.1-r1023/src

- Ejecutar make

  Make

Anexo 5: Ejecución de la topología en el ambiente virtual

Hasta el momento se ha instalado exitosamente la plataforma Mininet, el Controlador SDN que se va a utilizar, OpenDayLight, y se tiene el Script que simula la topología Fat-Tree. El siguiente paso es ejecutar la topología, anexarla con el Controlador y comprobar la conectividad entre todos los clientes.

Conectar la topología con el Controlador Opendaylight

De acuerdo a que opción se ha utilizado al descargar Mininet se realizará lo siguiente:

Opción 1 Descarga de Mininet VM

Si se ha descargado la imagen de Mininet para ejecutarse con Virtual Box, se debe colocar en red las VM de Mininet y Ubuntu, donde se encuentra instalado el controlador, de acuerdo a los siguientes pasos:

1) Seleccionar la VM de Mininet en Virtual Box, con click derecho, escoger la opción de Configuración.

Figura 17 Configuración VM Mininet. Por: Los autores

2) Seleccionar en el menú de lado izquierdo la pestaña de Red. El adaptador de red debe estar habilitado, conectado a: Adaptador puente, tal como se muestra en la imagen.

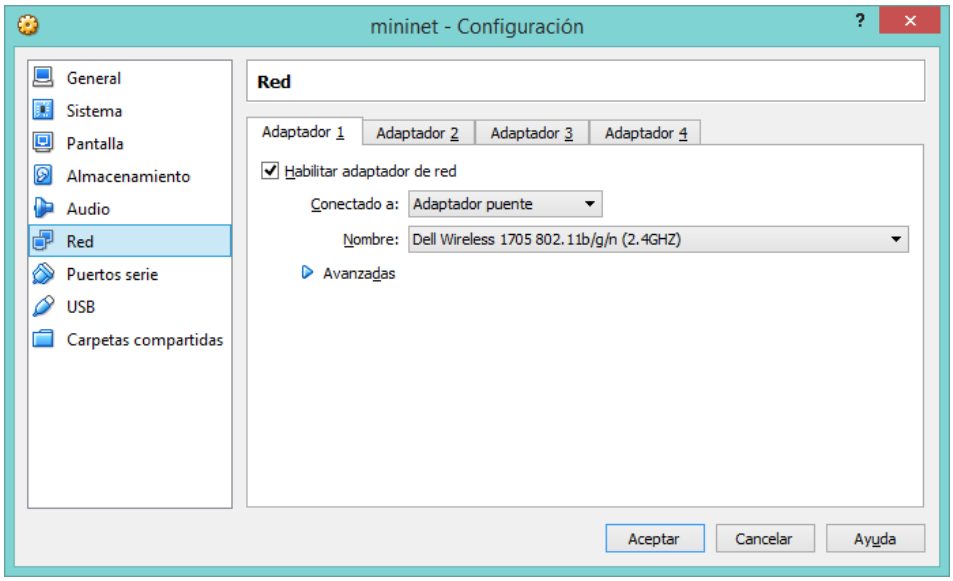

Figura 18 Opciones de Red VM Mininet. Por: Los autores

Se deben realizar los mismos pasos con la VM de Ubuntu, donde se encuentra instalado el Controlador.

3) Proceder a ejecutar las VM de Mininet y Ubuntu
4) Realizar la configuración de red para obtener respuesta de ping entre las 2 VMs.

   Se debe colocar información de ip, máscara de subred y Gateway por defecto que es el mismo Gateway de nuestra propia pc que se puede visualizar en las propiedades de red.

   VM MININET

   ifconfig eth0 192.168.0.7 netmask 255.255.255.0 up

   route add default gw 192.168.0.1

   VM Ubuntu

   ifconfig eth0 192.168.0.10 netmask 255.255.255.0 up

   route add default gw 192.168.0.1

5) Probar respuesta de ping al Gateway por defecto y entre las máquinas virtuales.

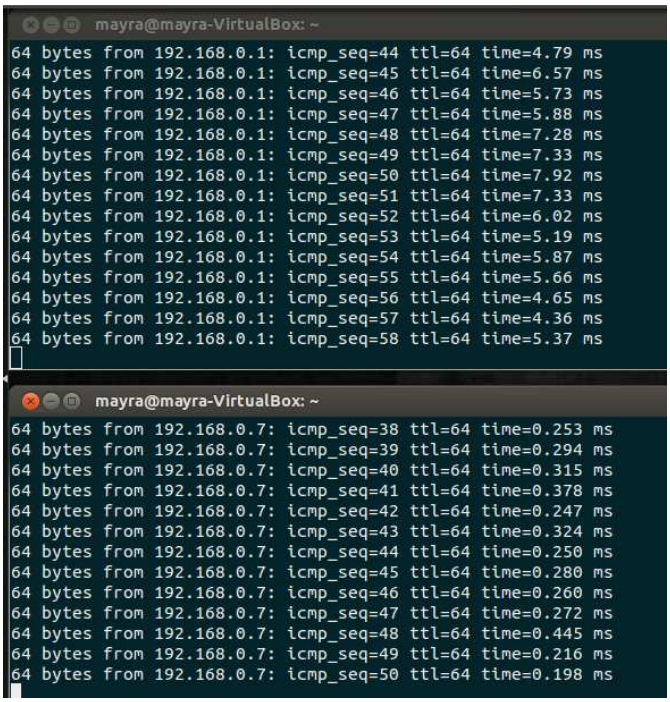

Figura 19 Máquina UBUNTU. Por: Los autores

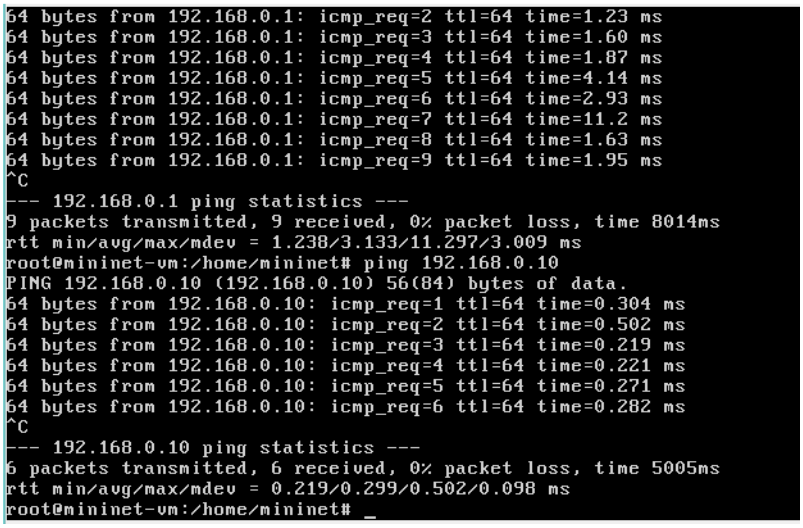

Figura 20 Máquina Mininet. Por: Los autores

6) Ya que la VM de Mininet corre bajo la distribución de Ubuntu Server, es algo incómodo manejarla de esta forma, por lo que se puede establecer una conexión ssh desde la máquina virtual de Ubuntu de acuerdo al siguiente comando:

ssh -XY mininet@192.168.0.7

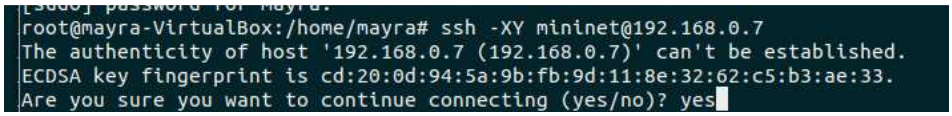

Figura 21 Acceso SSH desde VM Ubuntu a VM Mininet. Por: Los autores

Figura 22 Manejo de VM Mininet a través de VM Ubuntu. Por: Los autores

7) Una vez establecida la conexión hacia la VM de Mininet, se puede trabajar desde una sola máquina virtual con el Controlador y Mininet. Dentro de la máquina virtual de Mininet, en el directorio señalado se ha creado el Script de la topología Fat-tree:

   cd /home/usr/meshed-topo.py

   Se debe modificar el parámetro de conexión con el Controlador en el Script python. Para modificar el script se lo puede visualizar desde consola o a través del editor, según la preferencia utilizar los siguientes comandos:

   nano meshed-topo.py (para visualizarlo desde consola)
   gedit meshed-topo.py (para visualizarlo desde el editor)

   En información de Controller modificar y colocar la ip de la máquina virtual de Ubuntu, donde se encuentra el Controlador.

```
def createTopo():
    logging.debug("LV1 Create HugeTopo")
    topo = HugeTopo()
    topo.createTopo()
    topo.createLink()

    logging.debug("LV1 Start Mininet")
    CONTROLLER_IP = "192.168.0.10"
    CONTROLLER_PORT = 6633
    net = Mininet(topo=topo, link=TCLink, controller=None)
    net.addController( 'controller',controller=RemoteController,ip=CONTROLLER_IP,port=CONTROLLER_PORT)
    net.start()
```

Figura 23 Ejecución comando nano para visualizar script. Por: Los autores

De esta forma se realiza la conexión de la topología basada en Python con el Controlador OpenDayLight cuando se está trabajando con la plataforma de Mininet descargada como máquina virtual.

Opción 2 Descarga de Mininet por consola desde la fuente

Si se ha instalado la plataforma de Mininet a través de comandos desde una máquina virtual ya existente es mucho más fácil, ya que en la misma Máquina virtual de Ubuntu se encuentra instalado Mininet y OpenDayLight, así que por defecto se tiene conexión entre los dos.

Para esto en el Script de la topología Fat-Tree la información del controlador debe estar de la siguiente forma:

```
def createTopo():
    logging.debug("LV1 Create HugeTopo")
    topo = HugeTopo()
    topo.createTopo()
    topo.createLink()

    logging.debug("LV1 Start Mininet")
    CONTROLLER_IP = "127.0.0.1"
    CONTROLLER_PORT = 6633
    net = Mininet(topo=topo, link=TCLink, controller=None)
    net.addController( 'controller',controller=RemoteController,ip=CONTROLLER_IP,port=CONTROLLER_PORT)
    net.start()
```

Figura 24 Información de Controller SDN en script de topología. Por: Los autores

Una vez culminados estos pasos, se procede a ejecutar la topología y el controlador

Ejecutar el controlador.

Dentro de la máquina virtual de Ubuntu, donde previamente se ha instalado el controlador se debe ingresar al directorio:

cd/home/user/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight#

Luego en modo privilegiado o root ejecutar el siguiente comando:

export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64

./run.sh

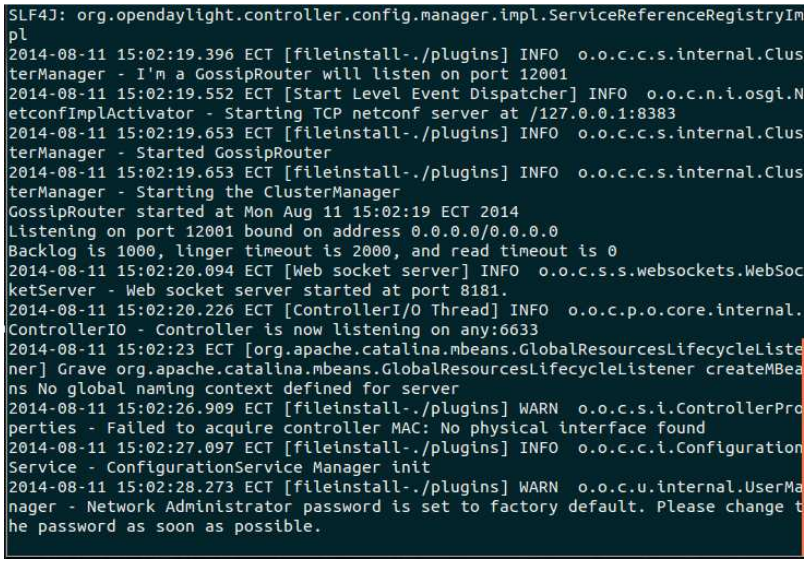

Figura 25 Ejecución del Controlador Opendaylight. Por: Los autores

Ahora el Controlador está operativo, y se debe establecer la conexión hacia la topología de Mininet.

Ejecutar el script de la topología Fat-Tree

Dentro del siguiente directorio

cd /home/usr/

Ejecutar el Script de la topología, para este caso el nombre del archivo es fattree.py

./meshed-topo.py

```
DEBUG:__main__:Class HugeTopo
DEBUG:root:LV1 Create HugeTopo
DEBUG:__main__:Class HugeTopo init
DEBUG:__main__:Start create Core Layer Swich
DEBUG:__main__:Create Core Layer
DEBUG:__main__:Start create Agg Layer Swich
DEBUG:__main__:Create Agg Layer
DEBUG:__main__:Start create Edge Layer Swich
DEBUG:__main__:Create Edge Layer
DEBUG:__main__:Start create Host
DEBUG:__main__:Create Host
DEBUG:__main__:Create Core to Agg
DEBUG:__main__:Create Agg to Edge
DEBUG:__main__:Create Edge to Host
DEBUG:root:LV1 Start Mininet
*** Creating network
*** Adding hosts:
4001 4002 4003 4004 4005 4006 4007 4008 4009 4010 4011 4012 4013 4014 4015 4016
*** Adding switches:
1001 1002 1003 1004 2001 2002 2003 2004 2005 2006 2007 2008 3001 3002 3003 3004 3005 3006 3007 3008
*** Adding links:
(1000.00Mbit) (1000.00Mbit) (1001, 2001) (1000.00Mbit) (1000.00Mbit) (1001, 2003) (1000.00Mbit) (1000.00Mbit) (1001, 2005) (1000.00Mbit) (1000.0
0Mbit) (1001, 2007) (1000.00Mbit) (1000.00Mbit) (1002, 2001) (1000.00Mbit) (1000.00Mbit) (1002, 2003) (1000.00Mbit) (1000.00Mbit) (1002, 2005) (
1000.00Mbit) (1000.00Mbit) (1002, 2007) (1000.00Mbit) (1000.00Mbit) (1003, 2002) (1000.00Mbit) (1000.00Mbit) (1003, 2004) (1000.00Mbit) (1000.00
Mbit) (1003, 2006) (1000.00Mbit) (1000.00Mblt) (1003, 2008)
```

Figura 26 Ejecución del script de topología Fat-Tree. Por: Los autores

Los siguientes comandos, ayudan a tener información de la red:

Mininet> net

Muestra las conexiones de cada nodo con el puerto respectivo.

```
mininet> net
4001 4001-eth0:3001-eth3
4002 4002-eth0:3001-eth4
4003 4003-eth0:3002-eth3
4004 4004-eth0:3002-eth4
4005 4005-eth0:3003-eth3
4006 4006-eth0:3003-eth4
4007 4007-eth0:3004-eth3
4008 4008-eth0:3004-eth4
4009 4009-eth0:3005-eth3
4010 4010-eth0:3005-eth4
4011 4011-eth0:3006-eth3
4012 4012-eth0:3006-eth4
4013 4013-eth0:3007-eth3
4014 4014-eth0:3007-eth4
4015 4015-eth0:3008-eth3
4016 4016-eth0:3008-eth4
1001 lo:  1001-eth1:2001-eth1 1001-eth2:2003-eth1 1001-eth3:2005-eth1 1001-eth4:2007-eth1
1002 lo:  1002-eth1:2001-eth2 1002-eth2:2003-eth2 1002-eth3:2005-eth2 1002-eth4:2007-eth2
1003 lo:  1003-eth1:2002-eth1 1003-eth2:2004-eth1 1003-eth3:2006-eth1 1003-eth4:2008-eth1
1004 lo:  1004-eth1:2002-eth2 1004-eth2:2004-eth2 1004-eth3:2006-eth2 1004-eth4:2008-eth2
2001 lo:  2001-eth1:1001-eth1 2001-eth2:1002-eth1 2001-eth3:3001-eth1 2001-eth4:3002-eth1
2002 lo:  2002-eth1:1003-eth1 2002-eth2:1004-eth1 2002-eth3:3001-eth2 2002-eth4:3002-eth2
2003 lo:  2003-eth1:1001-eth2 2003-eth2:1002-eth2 2003-eth3:3003-eth1 2003-eth4:3004-eth1
2004 lo:  2004-eth1:1003-eth2 2004-eth2:1004-eth2 2004-eth3:3003-eth2 2004-eth4:3004-eth2
2005 lo:  2005-eth1:1001-eth3 2005-eth2:1002-eth3 2005-eth3:3005-eth1 2005-eth4:3006-eth1
2006 lo:  2006-eth1:1003-eth3 2006-eth2:1004-eth3 2006-eth3:3005-eth2 2006-eth4:3006-eth2
2007 lo:  2007-eth1:1001-eth4 2007-eth2:1002-eth4 2007-eth3:3007-eth1 2007-eth4:3008-eth1
2008 lo:  2008-eth1:1003-eth4 2008-eth2:1004-eth4 2008-eth3:3007-eth2 2008-eth4:3008-eth2
3001 lo:  3001-eth1:2001-eth3 3001-eth2:2002-eth3 3001-eth3:4001-eth0 3001-eth4:4002-eth0
3002 lo:  3002-eth1:2001-eth4 3002-eth2:2002-eth4 3002-eth3:4003-eth0 3002-eth4:4004-eth0
3003 lo:  3003-eth1:2003-eth3 3003-eth2:2004-eth3 3003-eth3:4005-eth0 3003-eth4:4006-eth0
3004 lo:  3004-eth1:2003-eth4 3004-eth2:2004-eth4 3004-eth3:4007-eth0 3004-eth4:4008-eth0
3005 lo:  3005-eth1:2005-eth3 3005-eth2:2006-eth3 3005-eth3:4009-eth0 3005-eth4:4010-eth0
3006 lo:  3006-eth1:2005-eth4 3006-eth2:2006-eth4 3006-eth3:4011-eth0 3006-eth4:4012-eth0
3007 lo:  3007-eth1:2007-eth3 3007-eth2:2008-eth3 3007-eth3:4013-eth0 3007-eth4:4014-eth0
3008 lo:  3008-eth1:2007-eth4 3008-eth2:2008-eth4 3008-eth3:4015-eth0 3008-eth4:4016-eth0
controller
mininet>
```

Figura 27 Descripción de las conexiones en la topología Fat-Tree. Por: Los autores

Mininet> nodes

Muestra los hosts disponibles de la red.

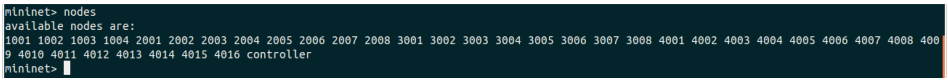

Figura 28 Detalle de los nodos de la red. Por: Los autores

Mininet> xterm "hostname"

Permite tener vista externa de un componente de la red.

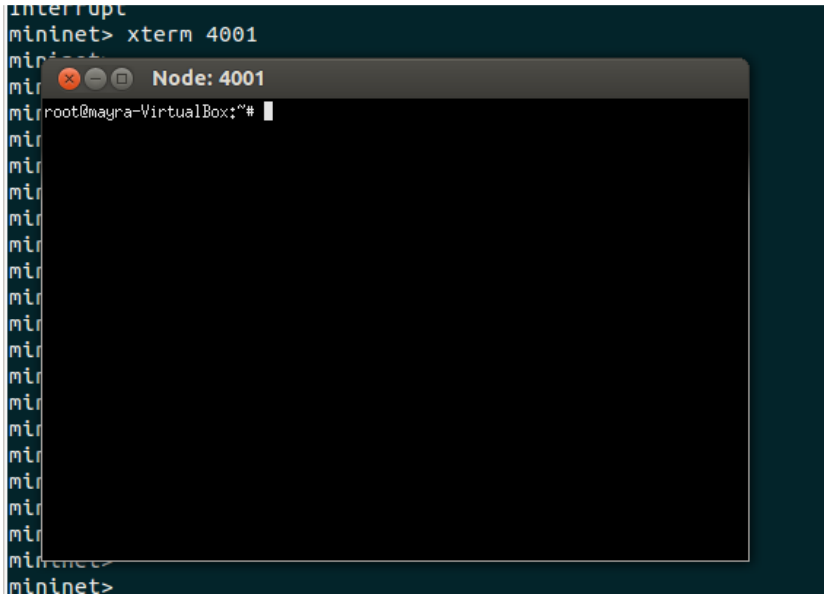

Figure 29 Ejecución de xterm. Por: Los autores

Probar conectividad entre hosts

La topología por sí sola no funciona, es necesaria la intervención del Controlador, para lograr la convergencia. Se puede visualizar si el controlador está realizando su trabajo haciendo un requerimiento de ping entre los hosts con el comando *pingall* que ejecuta una prueba de ping en modalidad mesh entre los hosts.

Mininet>pingall

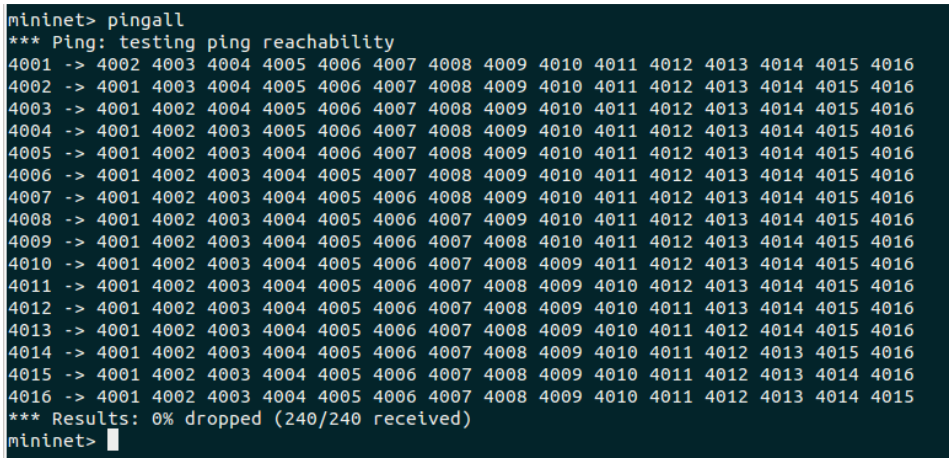

Figura 30 Respuesta de ping entre host tipo mesh. Por: Los autores

El test es exitoso, como se observa en la imagen. De esta forma se ha logrado crear un script basado en Python con una configuración de topología Fat-tree con ayuda de las librerías de Mininet y que sea administrado por un Controlador SDN (OpenDayLight), para emular un ambiente de redes definidas por software.

Anexo 5: Ejecución de test de generación de tráfico

Una vez que se ha establecido cada uno de los componentes que permitien evaluar la topología y presentar los resultados, para su realización, teniendo en cuenta que la topología está corriendo junto con el Controlador se ejecuta el script de test general que se encuentra basado en *bash*, de esta forma se pudo incorporar la función de *Randoms* que ayudarán a que el tráfico se genere con parámetros aleatorios.

Desde consola ejecutar en el directorio

cd /home/usr/

sudo bash ./test.sh

Podemos generar un pequeño archivo .*txt* donde se recopilará cada uno de los eventos que ocurran mientras se está ejecutando el test de esta manera:

sudo bash ./test.sh >>loggeneral.txt

Así también estos eventos no se mostrarán desordenadamente en la consola sino que se almacenan de forma ordenada, ayudando a identificar rápidamente si existe algún error en la ejecución.

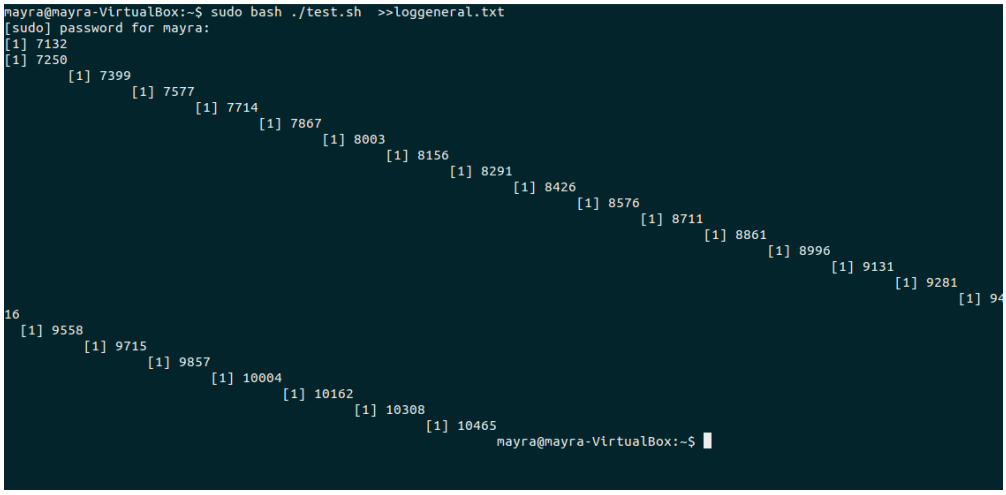

Figura 31 Ejecución de script de test general. Por: Los autores

Anexo 6: Script topología Fat-Tree

El modelo de programación que sirvió de referencia para establecer el diseño de la topología de red simulada, se obtuvo del repositorio de Scripts

https://gist.github.com/pichuang/9875468

El cual se modificó por los autores para establecer la topología deseada.

Para este caso esta configuración se almacenará en el directorio:

cd /home/usr/

Y el nombre del archivo es meshed-topo.py

Para crear el script se utiliza el siguiente comando:

nano meshed-topo.py

```
#!/usr/bin/python

"""
Custom topology for Mininet, generated by GraphML-Topo-to-Mininet-Network-Generator.
"""
from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import Link, Intf, TCLink
from mininet.topo import Topo
from mininet.util import dumpNodeConnections
from mininet.node import Node
from mininet.node import CPULimitedHost
import logging
```

```
import os
from mininet.util import dumpNodeConnections

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger( __name__ )

class BenchmarkTopo( Topo ):
    logger.debug("Class BenchmarkTopo")
    CoreSwitchList = []
    AggSwitchList = []
    EdgeSwitchList = []
    HostList = []
    iNUMBER = 0
    def __init__(self):
        logger.debug("Class HugeTopo init")
        iNUMBER = 4

        self.iNUMBER = iNUMBER
        self.iCoreLayerSwitch = iNUMBER
        self.iAggLayerSwitch = iNUMBER * 2
        self.iEdgeLayerSwitch = iNUMBER * 2
        self.iHost = self.iEdgeLayerSwitch * 2

        #Init Topo
        Topo.__init__(self)
    def createTopo(self):
        logger.debug("Start create Core Layer Swich")
        self.createCoreLayerSwitch(self.iCoreLayerSwitch)
        logger.debug("Start create Agg Layer Swich ")
        self.createAggLayerSwitch(self.iAggLayerSwitch)
        logger.debug("Start create Edge Layer Swich ")
        self.createEdgeLayerSwitch(self.iEdgeLayerSwitch)
        logger.debug("Start create Host")
        self.createHost(self.iHost)

            #Init Topo
        Topo.__init__(self)

    def createTopo(self):
        logger.debug("Start create Core Layer Swich")
        self.createCoreLayerSwitch(self.iCoreLayerSwitch)
        logger.debug("Start create Agg Layer Swich ")
        self.createAggLayerSwitch(self.iAggLayerSwitch)
        logger.debug("Start create Edge Layer Swich ")
        self.createEdgeLayerSwitch(self.iEdgeLayerSwitch)
```

```
    logger.debug("Start create Host")
    self.createHost(self.iHost)

"""
Create Switch and Host
"""

def createCoreLayerSwitch(self, NUMBER):
    logger.debug("Create Core Layer")
    for x in range(1, NUMBER+1):
        PREFIX = "100"
        if x >= int(10):
            PREFIX = "10"
        self.CoreSwitchList.append(self.addSwitch(PREFIX + str(x)))

def createAggLayerSwitch(self, NUMBER):
    logger.debug( "Create Agg Layer")
    for x in range(1, NUMBER+1):
        PREFIX = "200"
        if x >= int(10):
            PREFIX = "20"
        self.AggSwitchList.append(self.addSwitch(PREFIX + str(x)))

def createEdgeLayerSwitch(self, NUMBER):
    logger.debug("Create Edge Layer")
    for x in range(1, NUMBER+1):
        PREFIX = "300"
        if x >= int(10):
            PREFIX = "30"
        self.EdgeSwitchList.append(self.addSwitch(PREFIX + str(x)))

def createHost(self, NUMBER):
    logger.debug("Create Host")
    for x in range(1, NUMBER+1):
        PREFIX = "400"
        if x >= int(10):
            PREFIX = "40"
        self.HostList.append(self.addHost(PREFIX + str(x)))

"""
Create Link
"""
def createLink(self):
    logger.debug("Create Core to Agg")
    for x in range(0, self.iAggLayerSwitch, 2):
```

```
            self.addLink(self.CoreSwitchList[0], self.AggSwitchList[x], bw=10)
            self.addLink(self.CoreSwitchList[1], self.AggSwitchList[x], bw=10)
        for x in range(1, self.iAggLayerSwitch, 2):
            self.addLink(self.CoreSwitchList[2], self.AggSwitchList[x], bw=10)
            self.addLink(self.CoreSwitchList[3], self.AggSwitchList[x], bw=10)

        logger.debug("Create Agg to Edge")
        for x in range(0, self.iAggLayerSwitch, 2):
            self.addLink(self.AggSwitchList[x], self.EdgeSwitchList[x], bw=10)
            self.addLink(self.AggSwitchList[x], self.EdgeSwitchList[x+1], bw=10)
            self.addLink(self.AggSwitchList[x+1], self.EdgeSwitchList[x], bw=10)
            self.addLink(self.AggSwitchList[x+1], self.EdgeSwitchList[x+1], bw=10)

        logger.debug("Create Edge to Host")
        for x in range(0, self.iEdgeLayerSwitch):
            ## limit = 2 * x + 1
            self.addLink(self.EdgeSwitchList[x], self.HostList[2 * x], bw=10)
            self.addLink(self.EdgeSwitchList[x], self.HostList[2 * x + 1], bw=10)


topos = { 'benchmark': ( lambda: BenchmarkTopo() ) }

# here the code defining the topology ends
# the following code produces an executable script working with a remote controller
# and ssh access to the the mininet hosts from within the ubuntu vm

def setupNetwork():
    "Create network and run simple performance test"
    logging.debug("LV1 Create HugeTopo")
    topo = BenchmarkTopo()
    topo.createTopo()
    topo.createLink()

    # Controller for KTR network
    net = Mininet(topo=topo, controller=lambda a: RemoteController( a,
ip='127.0.0.1', port=6633 ), host=CPULimitedHost, link=TCLink)
    return net

def connectToRootNS( network, switch, ip, prefixLen, routes ):
#def connectToRootNS( network, ip, prefixLen, routes ):
    """Connect hosts to root namespace via switch. Starts network.
      network: Mininet() network object
      switch: switch to connect to root namespace
      ip: IP address for root namespace node
      prefixLen: IP address prefix length (e.g. 8, 16, 24)
```

```
    routes: host networks to route to"""
    # Create a node in root namespace and link to switch 0
    root = Node( 'root', inNamespace=False )
    #for switch in network.switches:
    intf = TCLink( root, switch ).intf1
    root.setIP( ip, prefixLen, intf )
    # Start network that now includes link to root namespace
    network.start()
    # Add routes from root ns to hosts
    for route in routes:
        root.cmd( 'route add -net ' + route + ' dev ' + str( intf ) )

def sshd( network, cmd='/usr/sbin/sshd', opts='-D' ):
    "Start a network, connect it to root ns, and run sshd on all hosts."
    ip = '10.0.2.15'  # our IP address on host network
    routes = [ '10.0.0.0/8' ]  # host networks to route to
    switch = network.switches[ 0 ]  # switch to use
    connectToRootNS( network, switch, ip, 8, routes )
    #connectToRootNS( network, ip, 8, routes )
    for host in network.hosts:
        host.cmd( cmd + ' ' + opts + '&' )
    print
    print "*** Hosts are running sshd at the following addresses:"
    print
    for host in network.hosts:
        print host.name, host.IP()
    print
    print "*** Type 'exit' or control-D to shut down network"

    print "Dumping host connections"
    dumpNodeConnections(network.hosts)
    print "Testing network connectivity"
    network.pingAll()

    CLI( network )
    for host in network.hosts:
        host.cmd( 'kill %' + cmd )
    network.stop()

if __name__ == '__main__':
    setLogLevel('info')
    #setLogLevel('debug')
    sshd( setupNetwork() )
```

Una vez creado el archivo se debe colocar lo siguiente desde la consola como usuario privilegiado

cd /home/usr/

chmod +X meshed-topo.py

Esto convertirá al script en un archivo ejecutable.

Anexo 7: Script de generación de tráfico

El script detallado a continuación fue modificado a partir del modelo encontrado en la web:

https://github.com/sjas/assessing-mininet

```
#!/bin/bash
echo GENERADOR DE TRAFICO
# corre con bash nombre.sh
echo '***** SERVIDOR LOG1'
bash -i -l -c 'sshpass -p user ssh -t -o StrictHostKeyChecking=no user@10.0.0.4
"/home/user/D-ITG/D-ITG-2.8.1-r1023/bin/ITGLog 0>/dev/null" &'
contrecv=1
while [ $contrecv -lt 17 ]; do
RECV=10.0.0.$contrecv
bash -i -l -c 'sshpass -p user ssh -t -o StrictHostKeyChecking=no user@'$RECV'
"/home/user/D-ITG/D-ITG-2.8.1-r1023/bin/ITGRecv 0>/dev/null" &'
echo $RECV
let contrecv=contrecv+1
done
contador=1
while [ $contador -lt 4 ]; do
numSEND=$(($RANDOM %15)+1));
numRECV=$(($RANDOM %15)+1));
if [ "$numSEND" = 4 ] || [ "$numRECV" = 4 ]; then
        echo $contador
        echo "IP SERVIDOR LOG"
        echo " "
else
        ipSEND=10.0.0.$numSEND
        ipRECV=10.0.0.$numRECV
        echo $contador
        echo $ipSEND $ipRECV>>parejas.txt
        echo RECV $contador
        echo SEND $contador
```

```
        echo ""
        bash -i -l -c 'sshpass -p user ssh -t -o StrictHostKeyChecking=no user@'$ipSEND' "\
        /home/user/ktr/ITGscripts/ITGSendUDP_cbr '$ipRECV' 10.0.0.4  '$contador' 60000 75 2560 0>/dev/null && \
        echo FINISHED" &'
        let contador=contador+1
fi
done
##PING PRUEBA
echo ""
#echo ***** RECV $contador
#echo ""
#bash -i -l -c 'sshpass -p user ssh -t -o StrictHostKeyChecking=no user@10.0.0.16 "/home/user/D-ITG/D-ITG-2.8.1-r1023/bin/ITGRecv 0>/dev/null" &'
echo SEND $contador
bash -i -l -c 'sshpass -p user ssh -t -o StrictHostKeyChecking=no user@10.0.0.5 "/home/user/D-ITG/D-ITG-2.8.1-r1023/bin/ITGSend -a 10.0.0.15 -t 60000 -E 1400 -c 512 -l log_udp_snd_test_cbr -L 10.0.0.4 -x log_udp_rcv_test_cbr -X 10.0.0.4 0>/dev/null" &
```

Anexo 8: Script para eliminación de procesos

```
#!/bin/bash

#corre con  bash

rm -rf /home/alan/Documents/MATLAB/Mininet/logs1/decoded_*

rm -rf /home/alan/log_udp_*

rm -rf /home/alan/itg-logs/decoded_*

rm -rf /home/alan/ktr/MATLAB/pl_*

rm -rf /home/alan/ktr/MATLAB/testsuite_*

rm -rf /home/alan/pr.txt

rm -rf /home/alan/parejas.txt

killall -9 ITGRecv

killall -9 ITGLog

echo BORRADO
```

Anexo 9: Script para levantamiento de interfaces de red

```
#!/bin/bash
capa=1
while [ $capa -le 3 ]; do
case "$capa" in
1)      echo CORE $capa
        switchnum=1
        while [ $switchnum -le 4 ]; do
                interfnum=1
                while [ $interfnum -le 4 ]; do
                        switch=$capa'00'$switchnum;
                        interf=eth$interfnum;
                        ifconfig $switch-$interf up
                        echo $switch-$interf #>>interfacesdown.txt
                        let interfnum=interfnum+1
                done
        let switchnum=switchnum+1
        done
        ;;
2)      echo AGGREGATION $capa
        switchnum=1
        while [ $switchnum -le 8 ]; do
                interfnum=1
                while [ $interfnum -le 4 ]; do
                        switch=$capa'00'$switchnum;
                        interf=eth$interfnum;
                        ifconfig $switch-$interf up
                        echo $switch-$interf #>>interfacesdown.txt
                        let interfnum=interfnum+1
                done
        let switchnum=switchnum+1
```

```
        done
        ;;
3)      echo EDGE $capa
        switchnum=1
        while [ $switchnum -le 8 ]; do
                interfnum=1
                while [ $interfnum -le 4 ]; do
                        switch=$capa'00'$switchnum;
                        interf=eth$interfnum;
                        ifconfig $switch-$interf up
                        echo $switch-$interf #>>interfacesdown.txt
                        let interfnum=interfnum+1
                done
        let switchnum=switchnum+1
        done
        ;;
esac
let capa=capa+1
done
rm -r /home/alan/interfacesdown.txt
rm -r /home/alan/sumaAC.txt
rm -r /home/alan/sumaEA.txt
rm -r /home/alan/sumaEH.txt
```