Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DESPLIEGUE Y EVALUACIÓN EN ENTORNOS VIRTUALIZADOS DE ESCENARIOS SDN CON CONTROLADORES MÚLTIPLES

TRABAJO FIN DE MÁSTER

Danellys Castillo Vejerano

2019

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DESPLIEGUE Y EVALUACIÓN EN ENTORNOS VIRTUALIZADOS DE ESCENARIOS SDN CON CONTROLADORES MÚLTIPLES

Autor

Danellys Castillo

Director

Francisco Javier Ruiz Piñar

Departamento de Ingeniería de Sistemas Telemáticos

2019

Resumen

Las redes de comunicación convencionales suelen ser complejas y de difícil administración, tienen una gran variedad de dispositivos, muchos de los cuales funcionan con software cerrado y de propietarios, además los protocolos de red requieren años de pruebas para su estandarización y portabilidad. Todos estos factores hacen que el proceso de innovación sea mucho más lento y aumenta los costos tanto operacionales como de implementación. De la necesidad de desarrollar un nuevo paradigma de redes que se adapte a las tendencias computacionales, a los requisitos de tráfico y gestión surgen las redes definidas por software o SDN (Software Defined Networking). [1]

Las redes definidas por software separan el plano de control del plano de datos, centraliza de forma lógica la inteligencia de la red, e incorporan la programabilidad del comportamiento de la red.

El plano de control de las SDN contiene uno o más controladores que son los encargados de determinar las acciones a realizar con el tráfico recibido y comunicárselo a los dispositivos de red. Existe una gran variedad de controladores de código abierto y de propietario. Este trabajo se enfoca en los controladores OpenDaylight y ONOS que son dos de los controladores de código abierto más utilizados.

La existencia de más de un controlador puede ser necesaria en entornos SDN por razones de fiabilidad, escalabilidad u organizativas. En este trabajo se presentará una herramienta creada para desplegar sistemas SDN virtualizados con múltiples controladores de software abierto que le permite al usuario ingresar diferentes parámetros para ejecutar y evaluar diferentes escenarios.

Palabras Clave: SDN, Redes Definidas por Software, OpenDaylight, VNX, OpenFlow, Sistemas Distribuidos.

Abstract

Conventional communication networks are often complex and difficult to manage, they have a great variety of devices, many of them work with proprietary and closed software, the network protocols require years of testing for standardization and portability. All these factors make the innovation process much slower and increases both operational and implementation costs. From the need to develop a new paradigm of networks that adapt to the computational tendencies, to the requirements of traffic and management arise the networks defined by software or SDN (Software Defined Networking).

Software-defined networks separate the control plane from the data plane and also logically centralize the intelligence of the network

The control plane of the SDN contains one or more controllers that are in charge of determining the actions to be carried out with the received traffic and communicating it to the network devices. There is a wide variety of open source and proprietary controllers. This work focuses on OpenDaylight and ONOS which are two of the most used open source controllers.

The existence of more than one controller may be necessary in SDN environments for reliability, scalability or organizational reasons. In this work we will present a tool created to deploy virtualized SDN systems with multiple open software drivers that allows the user to enter different parameters to execute and evaluate different scenarios.

Key Words: SDN, Software Defined Networking, OpenDaylight, VNX, OpenFlow, Distributed Systems.

Índice general

Índice de figuras

Siglas

API	Application Programming Interface
BGP	Border Gateway Protocol
CDPI	Control Data Plane Interface
CORD	Central Office Re-architected as a Datacenter
CPU	Central Processing Unit
EIGRP	Enhanced Interior Gateway Routing Protocol
ForCES	Forwarding and Control Element Separation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol
LXC	Linux Containers
MD-SAL	Model-driven Service Abstraction Layer
NBI	Northbound Interface
NFV	Network Function Virtualization
ODL	OpenDaylight
ONF	Open Networking Foundation
OSPF	Open Shortest Path First
OVS	Open vSwitch
PCE	Path Computation Element
REST	Representational State Transfer
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VNX	Virtual Network over Linux

1 Introducción

Las redes definidas por software o SDN surgen para solucionar inconvenientes de las redes convencionales como la complejidad en implementación y administración y la lentitud en los procesos de innovación, entre otros.

De acuerdo con la Open Networking Foundation (ONF), SDN se desarrolló basándose en tendencias computacionales claves como lo es el cambio de parámetros de tráfico tanto a nivel de centros de datos empresariales como a nivel de usuarios, por ejemplo, los usuarios acceden a aplicaciones desde cualquier dispositivo, cualquier lugar y en cualquier momento. Otra tendencia es que los usuarios usan cada vez más dispositivos personales para acceder a la red corporativa por lo que se necesita adaptar de manera precisa a los dispositivos para proteger los datos corporativos y la propiedad intelectual. El aumento de los servicios en la nube también está entre estas tendencias y es que proporcionar aprovisionamiento de autoservicio, ya sea en una nube privada o pública, requiere gran elasticidad de los recursos informáticos, de almacenamiento y de red. El manejo de datos masivos o "big data" que requiere gran ancho de banda también es una de estas tendencias computacionales.

Por otro lado, contar con una red de alta disponibilidad es de gran importancia para muchos de los servicios de las empresas. La alta disponibilidad busca asegurar cierto grado de continuidad operacional durante un periodo de tiempo; para servicios muy críticos las empresas buscan una disponibilidad de lo que llaman los cinco 9, es decir un 99.999%, que equivale a un máximo de 5.26 minutos de inactividad al año. En las redes definidas por software una manera de contar con alta disponibilidad es por medio de sistemas distribuidos de múltiples controladores.

1.1 Motivación

El tema de este trabajo fue elegido dado el gran impacto que están teniendo las redes definidas por software por la manera en que se adaptan a las crecientes necesidades del mercado.

En este trabajo se realiza un estudio y evaluación de despliegues de controladores múltiples controladores motivado por los beneficios adicionales que representa, como lo es la alta disponibilidad, ya que para la gran mayoría de los servicios de una empresa tener un sistema sin alta disponibilidad es algo obsoleto; siendo así, evaluar las prestaciones y funcionamiento resulta de gran importancia.

1.2 Objetivos

1.2.1 Objetivo General

El objetivo general de este trabajo es presentar el desarrollo de una herramienta que permite desplegar escenarios de múltiples controladores utilizando software abierto, permitiendo así evaluar las prestaciones y comparar la manera en que los sistemas distribuidos de múltiples controladores operan y bajo qué circunstancias se obtienen mejores prestaciones.

1.2.2 Objetivos Específicos

- Crear una herramienta con interfaz gráfica y que emplee software abierto.
- Desplegar entorno virtualizado con la cantidad de controladores, conmutadores y hosts indicados por el usuario.
- Formar clúster de controladores utilizando ONOS
- Formar clúster de controladores utilizando OpenDaylight.
- Realizar experimentos de escalabilidad, disponibilidad y rendimiento.

1.3 Estructura del Trabajo

El presente trabajo se encuentra estructurado de la siguiente manera:

- Capítulo 1 (Introducción): Se presenta los objetivos del trabajo y la razón por la cual fue elegido este tema.
- Capítulo 2 (Redes Definidas por Software): Se muestra información desde cómo surgieron las redes SDN, su arquitectura y beneficios, así como también el protocolo OpenFlow que es el más estandarizado.
- Capítulo 3 (Múltiples Controladores en SDN): Se explican los beneficios de contar con múltiples controladores en las redes definidas por software, así como también estrategias de diseño que emplean los diferentes controladores y las pros y contras de cada uno.
- Capítulo 4 (OpenDaylight): Se explica el funcionamiento y arquitectura de OpenDaylight que es uno de los controladores a emplear por la herramienta.
- Capítulo 5 (ONOS): Se presentan las principales funcionalidades y arquitectura de ONOS, el segundo de los controladores que utilizará la herramienta.
- Capítulo 6 (Diseño de la Herramienta): Se exponen los requisitos funcionales tomados en cuenta al momento de desarrollar la herramienta, así como también la manera en que éstos fueron implementados.
- Capítulo 7 (Implementación y Pruebas de la herramienta): En este capítulo se presentan las pruebas a realizar en la herramienta para validar su correcto funcionamiento.
- Capítulo 8 (Evaluación de Entornos SDN con Múltiples Controladores): Se presentan resultados del uso de la herramienta para diferentes escenarios.

- Capítulo 9 (Conclusiones y Trabajos Futuros): Se expresan conclusiones en base a los resultados obtenidos en el capítulo anterior y a las investigaciones realizadas respecto a los controladores: se indican también futuras líneas de trabajo.
- Repositorio: El código empleado en la herramienta se encuentra disponible en el siguiente enlace de GitHub https://github.com/Danellys/Multi-Controller-SDN.

2 Redes Definidas por Software

Durante este capítulo se busca presentar un marco histórico sobre las redes definidas por software. Se estudia su arquitectura para comprender así su funcionamiento y los múltiples beneficios que proporcionan.

2.1 Marco Histórico

En la década del 2000 el incremento el volumen de tráfico y requerimientos como rendimiento, fiabilidad y previsibilidad hicieron que los operadores de red buscasen mejorar funciones de red, como el control de las rutas usadas para el tráfico.

El aumento en las velocidades de los enlaces troncales hizo que proveedores de equipos implementaran la lógica de reenvío de paquetes directamente en el hardware, separada del software. Además, los proveedores de servicios de internet (ISP – Internet Service Provider) se esforzaban por gestionar el tamaño y el alcance crecientes de sus redes, y las demandas de una mayor fiabilidad y nuevos servicios (como las redes privadas virtuales). Paralelamente a estas tendencias, los servidores a menudo tenían más memoria y recursos de procesamiento que los procesadores del plano de control de un router implementados solo uno o dos años antes [1]. Todo esto llevó a innovaciones como ForCES: En 2004 el IETF (Internet Engineering Task Force) propuso la separación de los elementos de reenvío y control (ForCES - Forwarding and Control Element Separation); el grupo de trabajo ForCES también propuso una arquitectura complementaria SoftRouter; entre otros estándares propuestos por IEFT que separaban el control estaban "Linux Netlink como un Protocolo de Servicio IP" y una arquitectura basada en elementos de cálculo de rutas (PCE).

Estos primeros intentos no lograron ganar tracción dado que la comunidad de internet consideraba que la separación de los planos de control y de datos era arriesgada, debido a la posibilidad de una falla en el plano de control. Adicional a los proveedores les preocupaba que la creación de interfaces de programación de aplicaciones estándar (API) entre el plano de control y de datos daría lugar a una mayor competencia.

Los primeros usos de un software abierto para la separación del plano de control del plano de datos tienen sus orígenes en el proyecto Ethane en el departamento de ciencias computacionales de la Universidad de Stanford, el diseño simple de conmutador del proyecto Ethane llevó a la creación de OpenFlow. La primera API para OpenFlow fue creada en 2008 y ese mismo año se creó NOX, un sistema operativo para redes. [2]

En el año 2011 se crea Open Networking Foundation que es una organización sin ánimos de lucro que busca crear estándares para las redes definidas por software y promover su uso. [3]

2.2 Arquitectura

Las redes definidas por software son redes programables en las que se separa el plano de control (software) del plano de datos. Podemos describir los dispositivos de red como elementos que "hacen cosas" o que "piensan cómo hacer las cosas". Los dispositivos de red pueden reenviar el tráfico y/o pensar cómo reenviar el tráfico, en las redes convencionales realizan ambas funciones, por ejemplo, un router reenvía el tráfico y también calcula las rutas (OSPF, EIGRP, etc).

La idea de las redes definidas por software es separar la parte de "pensar" de la parte de "hacer". La parte de "pensar" (cálculo de rutas, filtrado de paquetes, etc.) de todos los dispositivos de red es trasladada a un punto lógicamente centralizado, llamado controlador. Los dispositivos de red se encargarán de procesar (reenviar paquetes, por ejemplo) y cuando reciban un tipo tráfico para el cual no cuentan con una regla de procesado podrán contactar al controlador para que este les diga qué hacer.

La arquitectura de una red definida por software de acuerdo con la Open Network Foundation es la siguiente se muestra en la figura 1 [4].

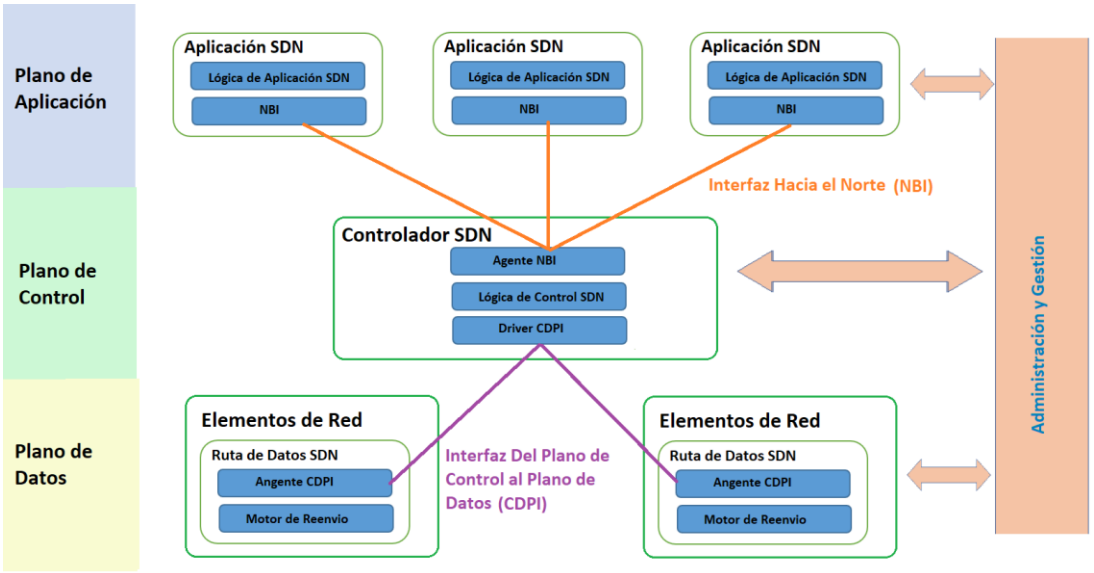

Figura 1. Arquitectura genérica de una SDN de acuerdo con la ONF. [4]

- Aplicaciones SDN: Son programas que pueden de manera directa y programable comunicar sus requerimientos y comportamientos de red deseados directamente al controlador SDN. Adicionalmente pueden utilizar una vista abstracta de la red para la toma de decisiones internas. La aplicación SDN está compuesta por la lógica de la aplicación SDN y uno o más controladores de dispositivos NBI (NorthBound Interface -Interfaz hacia el norte).
- Controlador SDN: Es una entidad lógicamente centralizada que se encarga de traducir los requerimientos de la capa de aplicación a la de ruta de datos,

proporcionando a la capa de aplicación una vista abstracta de la red que puede incluir estadísticas y eventos. El controlador SDN está compuesto por un agente NBI, la lógica del controlador SDN y un controlador de dispositivo CDPI (Control Data Path Interface) o interfaz sur del plano de control al plano de datos.

- Ruta de Datos SDN (SDN DataPath): Es un componente lógico de red que expone visibilidad y control sobre sus capacidades de reenvío y procesamiento de datos. La representación lógica, por lo tanto, puede abarcar todos o un subconjunto de los recursos físicos. Está compuesta por un agente CDPI y un conjunto de uno o más motores de reenvío de tráfico, también puede incluir, o no, funciones de procesamiento de tráfico. Es la infraestructura del plano de datos y se encarga de la conmutación de paquetes.
- Interfaz Sur o Interfaz CDPI: Interfaz definida entre el controlador SDN y la ruta de datos SDN, que proporciona control programable de todas las operaciones de reenvío, anuncio de capacidades, reporte estadístico y notificación de evento. Es posible que el CDPI se implemente en un entorno abierto, de manera interoperable e independientemente del proveedor y es este uno de los valores de las SDN.
- Interfaz norte (NBI): Son interfaces entre los controladores SDN y las aplicaciones SDN que proporcionan vistas abstractas del comportamiento de la red y sus requerimientos. Al igual que CDPI se espera que se implemente en un entorno abierto, interoperable e independiente del proveedor de dispositivo de red.
- Gestión y administración: El plano de administración cubre tareas estáticas que son mejor manejadas fuera de los planos de control, rutas de datos y aplicación, por ejemplo, asignación de recursos a los clientes, configuración de elementos físicos, etc.
- Controladores de medio y agente de interfaz: Es implementada por un par controlador de medio-agente, que representa el lado de la infraestructura y lado de cara a la aplicación. [4]

Uno de los principios claves de la arquitectura SDN es que las aplicaciones pueden estar informadas de la red en oposición a las redes tradicionales y otro de los principios básicos es que el plano de control es centralizado y desacoplado del plano de datos. Es importante resaltar que al decir centralizado no implica que el controlador esté físicamente centralizado; por razones de confiabilidad, escalabilidad y rendimiento el controlador puede estar lógicamente centralizado a través de un conjunto de controladores físicos.

2.3 OpenFlow

OpenFlow es la primera interfaz estandarizada y un protocolo comúnmente usado en redes definidas por software. Se encarga de la comunicación entre el controlador y el

conmutador. Fue desarrollado a partir de 2008 en la Universidad de Stanford a raíz del proyecto de investigación "OpenFlow: Enabling Innovation in Campus Networks" (OpenFlow: Habilitando Innovación en las redes del Campus).

Un conmutador OpenFlow consta de al menos tres componentes: Una tabla de flujo con las acciones asociadas a cada entrada para decirle al conmutador cómo procesar el flujo; un canal seguro que conecte el conmutador al controlador para permitir que los comandos y paquetes sean enviados entre ellos; y el protocolo OpenFlow que proporciona un estándar abierto y para la comunicación entre el conmutador (ruta de datos) y el controlador. [1]

Los conmutadores OpenFlow se pueden clasificar como conmutador OpenFlow dedicado y conmutador OpenFlow híbrido. El conmutador dedicado es aquel en el que el manejo del tráfico se basa sólo en las tablas de flujo que proporciona el controlador; por su parte el conmutador híbrido además de utilizar las tablas de flujo del controlador puede funcionar como un elemento de red convencional y debe tener un mecanismo para decidir si utilizará OpenFlow o no, pero esto se encuentra fuera del alcance de la especificación para OpenFlow.

En los conmutadores OpenFlow el tráfico es manejado por tablas de flujos. El controlador por medio del protocolo OpenFlow puede agregar, eliminar o modificar entradas en las tablas. Cuando el conmutador recibe un paquete, primero verifica en su tabla de flujo si encuentra alguna entrada que concuerde con el tipo de tráfico recibido. En caso afirmativo procede a enviar el tráfico por la interfaz indicada en la tabla de flujos o bien hacia otra tabla de flujos en caso de que se maneje múltiples tablas. Si el tipo de tráfico no coincide con ningún flujo en la tabla entonces se pueden tener diferentes comportamientos dependiendo de la configuración de la tabla que iría desde descartar los paquetes, enviarlos a otra tabla o enviarlos al controlador. En caso de que sean enviadas al controlador éste define un nuevo flujo para ese paquete y envía la entrada o entradas al conmutador para que sean añadidas a las tablas de flujo. Finalmente, el paquete se envía de vuelta al conmutador para ser procesado con las nuevas entradas creadas.

2.4 Controladores

Los controladores en una SDN son el sistema operativo de la red, quienes en conjunto con las aplicaciones de red deciden cómo se debe manejar el tráfico en la red y luego se lo comunican a los conmutadores

Existe una gran variedad de software de controladores tanto de código abierto como de propietario que difieren de acuerdo a las estrategias de diseño que cada una utiliza, entre los que podemos mencionar los siguientes [5]:

- OpenDaylight
- ONOS
- Project Calico
- The Fast Data Project
- Project Floodlight
- Beacon
- NOX/POX
- Open vSwitch
- vneio/sdnc
- Ryu Controller
- Cherry
- Faucet
- OpenContrail
- Nuage Virtualized Services Controller (VSC) de Nokia
- VortiQa Open Network Director de Freescale Semiconductor

2.5 Beneficios

Al lograr separar el plano de control del plano de datos y tener un sistema centralizado las redes definidas por software logran proporcionar múltiples beneficios como lo son:

- Reducción de costo de inversión: El hardware de los equipos de red puede ser básico reduciendo así los costes, ya que la inteligencia pasa al controlador, esto supone un gran ahorro económico.
- Reducción de costos operacionales: La gestión centralizada, eficiencia operativa y el mejor uso del hardware pueden reducir costos. [6]
- Reducir la complejidad por medio de automatización: Ofrece un marco de gestión y automatización de red flexible, permitiendo así desarrollar herramientas que automatizan muchas de las tareas de administración que se realizan manualmente en la actualidad. Estas herramientas de automatización pueden reducir la sobrecarga operacional, disminuir la inestabilidad de la red introducida por la posibilidad de error del operador. [7]
- Control centralizado: En los entornos con hardware de diferentes fabricantes permite una mejor administración ya que se puede utilizar un software de orquestación y herramientas de gestión para una rápida implementación, configuración y actualización de dispositivos; de este modo no se requiere tener grupos para la gestión de dispositivos según su fabricante.
- Mejor tasa de innovación: Permite acelerar la innovación del negocio, ya que el personal de IT puede programar y reprogramar los dispositivos de red mucho

más rápido, lo que permite adaptarse a las necesidades del negocio y requerimientos de los usuarios.

- Mayor confiabilidad y seguridad: Hace que sea más fácil responder de una manera más rápida y proactiva ante un ataque. Un ejemplo sería el caso denegación de servicios, el administrador de la red podría redirigir el tráfico mucho más rápido, previniendo así que se paralice la red. [8]
- Mejor Experiencia de usuario: En las redes definidas por software las aplicaciones pueden tener una vista abstracta de la red que les permite tomar decisiones internas.

Es importante tomar en cuenta que además de beneficios algunas características de SDN pueden suponer factores de riesgo que es importante mitigar, por ejemplo, el control centralizado supone también un blanco de ataques de seguridad.

3 Controladores Múltiples en SDN

En una red definida por software se cuentan con dos variantes básicas: un único controlador con los riesgos que ello conlleva o varios con controladores distribuidos. Para determinar el diseño que se desea tener en un entorno de múltiples controladores hay diferentes aspectos a considerar y que se analizan en este capítulo; se presentan también los beneficios que proporciona este tipo de entorno.

3.1 Beneficios

Entre los beneficios que proporcionan los entornos de múltiples controladores en las redes definidas por software tenemos:

- Escalabilidad: En las redes con un único controlador es más limitada la cantidad de solicitudes procedentes de los conmutadores que se pueden manejar, para solucionar este inconveniente se intenta limitar las solicitudes de rutas enviadas por los conmutadores, sin embargo, esto contradice lo propuesto por las SDN ya que requiere implementar inteligencia a nivel de los conmutadores. Por su parte en los entornos con varios controladores es posible manejar mayores solicitudes y aumentar el número de controladores si así fuese necesario. [9]
- Alta Disponibilidad: El tener un entorno con un controlador es sinónimo de un único punto de fallo, ya que si falla los conmutadores no podrán enviar solicitudes sobre cómo deben encaminar los paquetes y la red dejará de operar en su totalidad cuando las rutas expiren. Como alternativa para resolver este inconveniente los conmutadores pueden operar con OpenFlow Híbrido de modo que ante estas situaciones se transforman en conmutadores Ethernet tradicionales o de capa 3, aunque esto reduce uno de beneficios de una SDN que implementar hardware de bajo coste. Al contar con múltiples controladores se aumenta la tolerancia a fallo del sistema, ya que si una de ellas falla otra puede realizar sus funciones.

3.2 Análisis de Diseño

Al momento de diseñar una SDN con controladores físicamente distribuidos se debe tener presente que existen diferentes tipos de estrategias de diseño y que cada una de estas proporcionan mayor o menor ventaja en parámetros como escalabilidad, tolerancia a falla, consistencia de los datos y privacidad. En [10] Yustus Eko nos presentan una clasificación de controladores de acuerdo con diferentes estrategias de diseño.

3.2.1 Estrategias de Conexión del Conmutador al Controlador

Como métodos de conexión entre el conmutador y el controlador en sistema centralizado se tienen dos tipos: esclavo/maestro y IP alias.

En una conexión esclavo/maestro el conmutador conoce las direcciones IP de todos los controladores y es capaz de comunicarse con cada uno de ellos. A partir de la versión 1.2 de OpenFlow se soportan tres tipos de roles los controladores que son master, slave y equal. Un conmutador puede estar conectado a varios controladores con el rol de slave o equal, pero solo a un master [11]. El máster es quien tiene mayor prioridad y puede realizar operaciones de lectura y escritura en el conmutador, un controlador con el rol de equal también puede realizar operaciones de lectura y escritura, por su parte un controlador slave únicamente realiza operaciones de lectura.

Por otro lado, en una conexión IP alias se asocian varias direcciones IP a una sola interfaz. En [12] nos presentan un controlador que utiliza una conexión IP alias, que cuenta con un conjunto de direcciones IP para controladores que tiene tantas direcciones IP como conmutadores existan en la red y esas direcciones IP de controladores se dividen entre los controladores existentes. Por ejemplo, si en una red se tienen 15 conmutadores y 3 controladores, cada controlador podría tener asignado 5 direcciones IP. En caso de fallo de uno de los controladores las direcciones IP se asignan a otros controladores. Esta técnica permite que, en caso de fallo, el conmutador continúe con conexión sin realizar ningún cambio en su configuración.

A nivel de escalabilidad los mecanismos de migración están mayormente basados en los roles de esclavo/maestro de OpenFlow por lo que son más fiables. En cuanto a consistencia en la información el modelo de Esclavo/Maestro presenta desventaja ya que no sólo el conmutador Maestro tendrá permiso de escritura, sino también los que tengan el rol de Equal, pudiendo así crear conflicto de reglas en un mismo conmutador; el modelo IP alias no tiene este inconveniente ya que el conmutador se comunica con un solo controlador. En cuanto a tolerancia a fallas ambas estrategias presentan mecanismos que permiten una conexión dinámica entre conmutadores y controladores en caso de ser necesario.

3.2.2 Estrategias de Distribución de Información

A fin de proporcionar un control lógicamente centralizado en un entorno SDN físicamente distribuido se requiere que al menos uno de los controladores conozca el estado global de la red, para lograr esto es necesario compartir y distribuir la información del estado de red entre los controladores. Analizaremos dos métodos para la distribución de la información de red: modelo jerárquico y modelo plano.

En un modelo jerárquico, también conocido como modelo vertical, el plano de control está formado por más de un nivel, los controladores en el nivel más bajo controlan ciertos dominios de la red, mientras que el controlador o controladores del nivel más alto tienen una versión global de la red. Los controladores de niveles superiores (root) proporcionan a los controladores de niveles inferiores (locales) conectividad con el resto de la red.

En el modelo plano u horizontal todos los controladores que conforman el clúster tienen una visión global de la red, para que se construya esta visión se debe recibir el estado local de cada uno de los controladores y a partir de allí se debe informar al resto de cualquier cambio en su dominio.

A nivel de escalabilidad, según estudios presentados [13], el modelo jerárquico proporciona mejores prestaciones que un modelo plano en el que cada controlador tenga una visión global de la red. En cuanto a tolerancia a falla, suele ser mejor un modelo plano, ya que en un modelo jerárquico un controlador root no podría ser reemplazado por un controlador local y viceversa, reduciendo así la cantidad de controladores disponibles en caso de falla. Por otro lado, el modelo jerárquico presenta ventajas sobre el modelo plano en cuanto a escalabilidad, ya que menos controladores contienen el estado global de la red.

3.2.3 Estrategias de Coordinación entre Controladores

El hecho de tener múltiples controladores en una red SDN puede causar inconsistencia. Por ejemplo, si el controlador A quiere realizar un cambio en la ruta de un conmutador que está bajo el dominio del controlador C, pero un controlador B quiere eliminar esa ruta, entonces el controlador C podría tener problemas en decidir cuál operación se debe ejecutar si no se coordinan de manera adecuada.

En las SDN se utilizan algoritmos de consenso que puede estar basado en líder o sin líder. En las SDN sin líder cada controlador tiene el mismo privilegio y puede contactar directamente a otro controlador para programar la red arbitrariamente. Por otro lado, cuando se utiliza una estrategia basada en líder éste debe asegurarse de que todos los controladores se comportan como uno solo y toda solicitud de cambio en la red debe ser enviada antes al líder para que verifique si pasa el consenso.

En cuanto a escalabilidad, un diseño con líder tiende a ser más lento, ya que antes de realizar cualquier cambio requiere que se valide el consenso. Sin embargo, de acuerdo con algunos estudios [15], resulta ser más escalable cuando se tienen muchos nodos. En lo referente a tolerancia a falla, el tiempo de recuperación después que el líder falla es mayor, ya que se debe realizar un proceso de elección de líder. Por otra parte, es muy importante considerar también que en un entorno sin líder es más probable que se presente inconsistencia.

3.2.4 Estrategias de Conexión de Administración

De acuerdo a la especificación de OpenFlow, la configuración de la interfaz sur se puede manejar por medio de dos tipos de conexiones: In-band y Out-band. En una conexión Out-band se cuenta con un enlace dedicado para intercambiar mensajes de coordinación entre los dispositivos de red y los controladores; mientras que en una conexión In-band no se tiene un enlace dedicado y se utilizan los mismos enlaces que transportan el tráfico de datos de la red.

Al analizar las ventajas y desventajas de estos tipos de conexión se tiene que a nivel de escalabilidad y privacidad una conexión Out-band presenta mayores beneficios, en cuanto a escalabilidad porque no agrega carga al tráfico de datos que afecte las prestaciones de la red, y en privacidad porque al no compartir el enlace es menos probable que algún atacante intercepte mensajes de configuración. En lo referente a consistencia, según estudios realizados se encontró que las redes con conexión out-band presenta menos resistencia cuando se sufre de una partición en la red, esto basado en un escenario en el que el controlador no pueda actualizar la vista de la topología de la red. [14]

3.2.5 Clasificación de Controladores SDN Distribuidos

En [10] nos presentan una clasificación de los proyectos de controladores distribuidos en base a las estrategias de diseño como se muestran en la tabla 1.

Tabla 1. Clasificación de Controladores SDN distribuidos [10]

Proyectos de Controladores SDN Distribuidos	**Proyectos SDN relacionados**	**Estrategias de Conexión del Conmutador al Controlador**	**Estrategias de Distribución de Información**	**Estrategias de Coordinación entre Controladores**
Easte/West Bridge	Foodlight, NOX		´Plano	Sin Líder
Elasticon		Esclavo/Maestro	Plano	Con Líder
DISCO	Foodlight		Plano	Sin Líder
HyperFlow	NOX		Plano	Sin Líder
OpenDaylight		Esclavo/Maestro	Plano	Con Líder
ONOS		Esclavo/Maestro	Plano	Con Líder
D-SDN		Esclavo/Maestro	Jerárquico	Con Líder
IRIS	Foodlight, Beacon	Esclavo/Maestro	Jerárquico	Con Líder

ONIX	NOX		Plano	Con Líder
Kandoo		Esclavo/Maestro	Jerárquico	Con Líder
Controlling SDN		IP Alias	Plano	Con Líder
FlowBroker	Floodlight		Jerárquico	Con Líder
Ravana	Ryu	Esclavo/Maestro	Plano	Con Líder

4 OpenDaylight

OpenDaylight es un proyecto SDN de código abierto que surgió en el 2013 alojado por The Linux Foundation [15]. El controlador OpenDaylight es uno de los más utilizados y se define como altamente disponible, escalable, extensible, modular y de múltiples protocolos; soporta aplicaciones como lo son:

- Entrega de Servicios Automatizadas
- Cloud y NFV
- Optimización de Recursos de Red
- Visualización y Control

En este capítulo se presenta la arquitectura y funcionamiento de OpenDaylight,se incluye una guía sobre el proceso de instalación y formación de un clúster.

4.1 Arquitectura

La estructura básica de un controlador ODL se muestra en la figura 2.

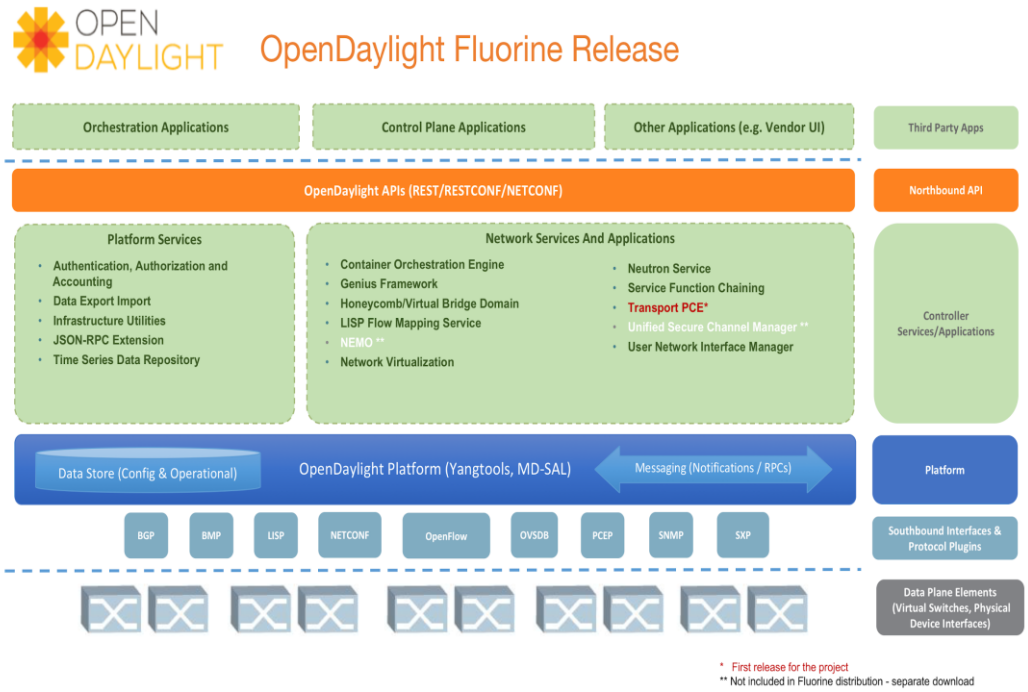

Figura 2. Arquitectura del controlador OpenDaylight [15].

- **Controlador de Servicios y Aplicaciones:** Como se muestra en la figura 2, esta capa está dividida en dos módulos, una plataforma de servicios y los servicios y aplicaciones de red.

  El módulo plataforma de servicios proporciona servicios específicos como el de autenticación, autorización y contabilización que permite al usuario autenticarse por medio de credenciales. Luego verifica cuáles operaciones tiene permitido realizar y luego registra cada una de esas operaciones. Otra de las funciones que brinda este módulo es la de un repositorio de serie de datos temporales que sondea periódicamente el almacenamiento de datos operativos de OpenDaylight y por medio de la interfaz norte ponen estos datos disponibles a aplicaciones externas. [18]

Por su parte, el módulo de servicios y aplicaciones de red maneja funciones como la de virtualización de redes y el administrador de la interfaz de red de usuario que permite a las aplicaciones de software y orquestadores de servicios configurar y suministrar servicios de conectividad por medio de los modelos de datos y API que proporciona. [18]

- **Capa de Abstracción de Servicios:** El núcleo del controlador OpenDaylight es la capa de abstracción de servicios dirigida por modelos, conocida como MD-SAL por sus siglas en inglés (Model Driven -Service Abstraction Layer). Es quien permite a la interfaz sur soportar múltiples protocolos y donde interactúan los objetos y modelos que representan a los elementos de red y aplicaciones. La capa de abstracción de servicios oculta la complejidad de las API utilizando modelos YANG y permite interacciones por roles productor/consumidor; los productores son quienes implementan una API o los datos de una API y los consumidores son quienes utilizan esos datos. Un productor puede colocar datos en el almacenamiento de la capa de abstracción de servicios, mientras que un consumidor puede leer esos datos. [19]
- **Interfaz Norte:** Expone protocolos como NETCONF, REST y RESTCONF.
- **Interfaz Sur:** Se comunica con los elementos de red y abarca múltiples protocolos y plugin, como lo son OpenFlow, NETCONF, BGP, SNMP, LISP, etc.

4.2 Versiones

Desde su fundación en el 2013 el proyecto de OpenDaylight ha lanzado varias versiones de su controlador cuyos nombres son elementos químicos en orden de su número atómico.

Tabla 2. Versiones de OpenDaylight y sus principales cambios

Versión	**Fecha de Lanzamiento**	**Principales cambios**
Hydrogen	Febrero de 2014	Primera versión de OpenDaylight
Helium	Octubre de	Permite la creación de clúster y la posibilidad de

	2014	autenticación, autorización y contabilización.
Lithium	Junio de 2015	Se agrega el protocolo NETCONF a la interfaz northbound. En DLUX(interfaz web gráfica) se agrega un visualizador de topología de MDSAL.
Beryllium	Febrero de 2016	Se agrega nuevo plugin BGP Monitoring Protocol a la interfaz sur de OpenDaylight.
Boron	Noviembre de 2016	Soporte del protocolo CAPWAP para el aprovisionamiento de puntos de acceso inalámbricos.
Carbon	Junio de 2017	Se agrega el plugin IoTDM para permitir la fácil implementación de nuevos plugin de dispositivos.
Nitrogen	Septiembre de 2017	Se implementa karaf 4 que es el componente que permite seleccionar los protocolos y servicios que soporta el controlador, la nueva versión de karaf permite implementar características más rápido.
Oxygen	Marzo de 2018	Se introduce un plugin para el lenguaje P4. En el motor de orquestación de contenedores se introduce un plugin para kubernetes.
Fluorine	Agosto de 2018	Se agrega el plugin Transport PCEpara soportar una infraestructura de transporte óptico

4.3 Funcionamiento del Clúster

En OpenDaylight se utiliza para almacenar los estados de red un Distributed-DataStore. Si no se realiza ninguna configuración extra el estado de la red se almacena en un default shard (fragmento por defecto). OpenDaylight utiliza el algoritmo de consenso RAFT para sincronizar réplicas entre los miembros del clúster, los controladores del clúster hacia los cuáles se repliquen son los especificados en el archivo module-shards.conf (por defecto todos los miembros tendrán réplicas). [16] El algoritmo de Raft proporciona alta consistencia, pero al costo de una disminución en las prestaciones de lectura/escritura. Si bien el estado de la red puede estar en diferentes miembros del clúster, existirá un controlador líder y el resto serán followers, sólo el controlador líder será capaz de aceptar operaciones de lectura/escritura. Por otro lado, en cuanto a las conexiones esclavo/maestro OpenFlow intercambia mensajes de sincronización con el controlador maestro, además de aceptar mensajes controller-to-switch; al controlador esclavo sólo le permite leer su configuración. OpenDaylight asigna nuevos maestros a los dispositivos que hayan perdido la conexión con su controlador maestro. El controlador elegido como nuevo maestro le envía un

mensaje de solicitud de rol al dispositivo y recibirá una respuesta del dispositivo en caso de ser satisfactorio.

Para analizar mejor el funcionamiento se muestra la figura 3. En este caso tenemos un clúster de tres controladores y el controlador ODL1 contiene la réplica líder para todas las topologías. Supongamos que se necesita realizar un cambio en la topología C, si bien el máster de la topología C es ODL3 dicho controlador no podrá modificar la topología C, tendrá que comunicarse con ODL1 para solicitar que realice la operación de escritura, una vez ODL1 recibe dicha solicitud consulta el consentimiento a los otros controladores y hará efectivo el cambio cuando tenga mayoría de votos (dos en este caso).

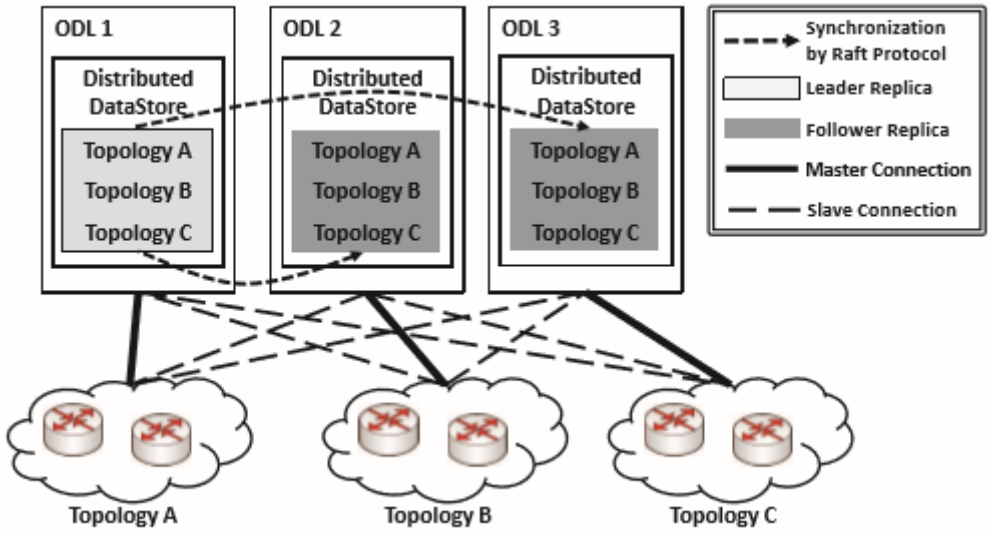

Figura 3. Funcionamiento de un clúster de controladores OpenDaylight [16]

4.4 Instalación y Formación de Clúster

A continuación se de detallan los comandos para la instalación y formación de un cluster de OpenDaylight en distribuciones de tipo Debian/Ubuntu.

Uno de los requisitos para el funcionamiento de ODL es el soporte a Java, para esto se realiza una actualización y luego se instala Java JDK y JRE como se muestra a continuación:

```
sudo apt-get update
sudo apt-get install default-jre
sudo apt-get install default-jdk
```

Una vez instalado Java se procede con la instalación, primero se crea una carpeta llamada "sdn" y sobre esta se descarga OpenDaylight (versión Nitrógeno en este caso), se procede a descomprimir y finalmente se inicia OpenDaylight. Los comandos utilizados y una captura de la inicialización de OpenDaylight se muestra a continuación.

```
mkdir sdn
cd sdn
sudo wget -c
https://nexus.OpenDaylight.org/content/repositories/public/org/OpenDaylight/integration/karaf/0.7.2/karaf-0.7.2.zip
unzip karaf-0.7.2.zip
cd  karaf-0.7.2/bin
./karaf
```

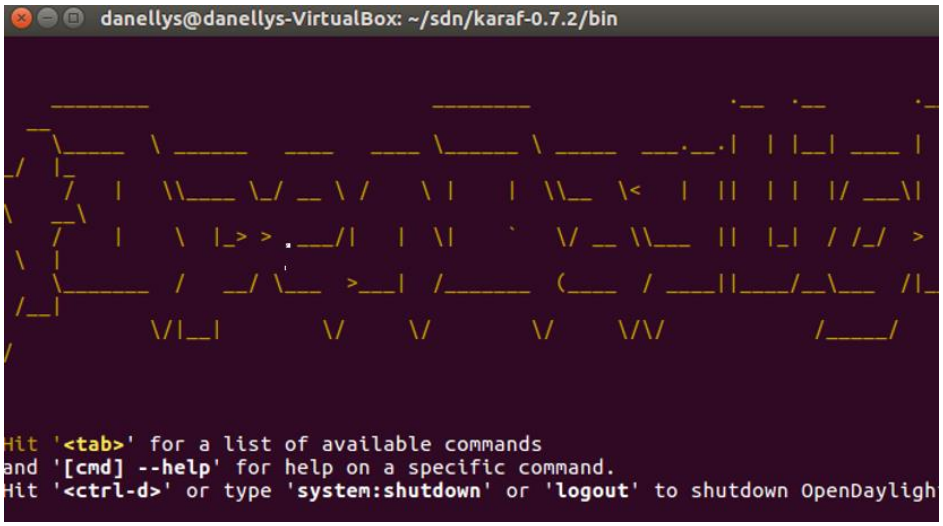

Figura 4. Consola Karaf de OpenDaylight en ejecución

Una vez OpenDaylight se encuentra en ejecución, se puede proceder con la instalación de los módulos que permiten el reenvío de tráfico, el funcionamiento del clúster y el acceso a la interfaz web. El comando que se debe emplear es el siguiente:

```
OpenDaylight-user@root>feature:install odl-mdsal-clustering
odl-l2switch-switch-ui   odl-dlux-core   odl-dluxapps-nodes
odl-dluxapps-topology odl-jolokia
```

Una vez se cuentan con las máquinas que tienen instalado OpenDaylight para proceder con la creación del clúster se utiliza un script *"configure_cluster.sh"* que se encuentra en la carpeta *"/sdn/karaf-0.7.2/bin"*. Este script es proporcionado por OpenDaylight para facilitar y agilizar la creación de clúster.

Para la utilización de este script se debe indicar primero un índice para el miembro que se está configurando, este índice debe ser un valor numérico entre 1 y N, donde N es la cantidad de miembros del clúster. A continuación, se debe colocar la dirección IP para cada uno de los otros miembros, la posición de la dirección IP del controlador a configurar debe corresponder con el índice. Por ejemplo, en un clúster de tres

controladores con direcciones IP 10.10.1.2, 10.10.1.3 y 10.10.1.4 se deberían ejecutar los siguientes comandos:

Miembro 1 (Dirección IP 10.10.1.2)

```
>> configure_cluster.sh 1 10.10.1.2 10.10.1.3 10.10.1.4
```

Miembro 2 (Dirección IP 10.10.1.3)

```
>> configure_cluster.sh 2 10.10.1.2 10.10.1.3 10.10.1.4
```

Miembro 3 (Dirección IP 10.10.1.4)

```
>> configure_cluster.sh 3 10.10.1.2 10.10.1.3 10.10.1.4
```

Al ejecutar estos comandos se modificarán los archivos *"module-shards.conf"* y *"akka.conf"* que se encuentran en la carpeta *"/sdn/karaf-0.7.2//configuration/initial/"*.En *"module-shards.conf"* podemos notar ahora que para cada shard se encuentran réplicas en cada uno de los miembros del clúster.

```
module-shards                          =                          [
    {
        name                       =                      "default"
        shards                          =                          [
            {
                name="default"
                replicas                    =                     [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
```

Por su parte el archivo *"akka.conf"* contiene la configuración de las direcciones IP que forman parte del clúster, el índice, entre otros parámetros de configuración como el tiempo transcurrido después que un nodo está inalcanzable para sacarlo del clúster.

```
cluster {
      seed-nodes=              ["akka.tcp://OpenDaylight-cluster-
      data@10.10.1.2:2550",
                               "akka.tcp://OpenDaylight-cluster-
      data@10.10.1.3:2550",
                               "akka.tcp://OpenDaylight-cluster-
      data@10.10.1.4:2550"]

       auto-down-unreachable-after             =             10s

       roles                              =                          [
              "member-2"
            ]
```

Por otro lado, los conmutadores de Open vSwitch deben conectarse a todos los nodos del clúster para brindar alta disponibilidad, al escribir el comando "ovs-vsctl show" podemos ver los conmutadores creados y sus conexiones con los controladores.

4.5 Seguridad en OpenDaylight

Al manejar un control centralizado los controladores se convierten en foco de ataques de seguridad, por lo que a fin de reducir vulnerabilidades OpenDaylight realiza una serie de recomendaciones básicas [17].

- **Recomendaciones de Implementación:** Cambiar las contraseñas por defecto; implementar en una red privada que no tenga acceso desde internet; separar la red de datos de la red de administración; crear políticas de autenticación tanto para los dispositivos que se conectan a la red de datos como para los que se conectan a la red de administración.
- **Asegurar la Interfaz Sur:** Los plugin para la interfaz sur de OpenDaylight tienen mecanismo de seguridad que es importante investigar cuando se están implementando para conectarse a los elementos de red. OpenFlow, por ejemplo, soporta conexiones TLS (Transport Layer Security) bidireccional. NETCONF por su parte, soporta conexión sobre SSH.
- **Asegurar RESTCONF utilizando HTTPS:** Se recomienda configurar un servidor Jetty para utilizar SSL (capa de puertos seguros). Los pasos detallados para su configuración se pueden encontrar en la web de OpenDaylight[16].
- **Seguridad en Clúster:** En las versiones actuales de OpenDaylight los mensajes que se intercambian entre nodos no están encriptados ni son autenticados, por lo que cualquiera con acceso a la red de administración donde se intercambian estos mensajes puede leer o crear mensajes.

4.6 Módulos Utilizados en OpenDaylight

En las sección 4.3 se mencionaron los módulos de OpenDaylight que son necesarios en la instalación. En este apartado se explica la función de cada uno de ellos:

- Odl-mdsal-clustering: Este módulo es el encargado de permitir el funcionamiento del clúster de controladores.
- Odl-l2switch-switch-ui: Este es el módulo que permite la conectividad en la red, permite aprender direcciones MAC e IP, también elimina bucles por lo que no es necesario instalar módulos adicionales de STP. Permite la instalación de flujos en cada conmutador basado en parámetros de tráfico en la red.
- Odl-dlux (core, apps-nodes, apps-topology): Estos módulos permiten utilizar la interfaz gráfica de ODL. Con las aplicaciones de nodos "odl-dluxapps-nodes" se podrán observar un listado de los dispositivos (conmutadores) detectados, así como sus flujos y conexiones. En la parte de topología se mostrará una representación gráfica de la red.
- Odl-jolokia: Este módulo permite el acceso tipo REST a JMX con JSON a través de HTTP. Este módulo nos permite realizar consultas para verificar si el clúster

se ha formado correctamente y que rol tienen los controladores. Para utilizarlo se puede escribir desde el navegador web la línea que se muestra a continuación.

```
http://<host>:8181/jolokia/read/org.opendaylight.controller
:Category=Shards,name=member-1-shard-inventory-
config,type=DistributedConfigDatastore
```

4.7 Incidencias en el Uso de OpenDaylight

Al momento de realizar pruebas con el clúster de Opendaylight se pudo observar inconvenientes para formar el clúster si todos los controladores se iniciaban al mismo tiempo. Esto se pudo observar al ingresar las consultas por medio del navegador web para verificar el rol de cada controlador.

5 ONOS

ONOS por las siglas en inglés para Sistema Operativo de Redes Abiertas (Open Networking Operative System) es un proyecto SDN de código abierto para controladores, este proyecto fue iniciado en diciembre de 2014 por el ON.Lab (Open Networking Lab) en asociación con otros empresas como NTT Communications y AT&T. En octubre de 2015 ONOS se unió a The Linux Foundation.

El objetivo de ONOS es proporcionar el plano de control para una red definida por software, gestionar componentes de red, ejecutar programas o módulos de software para proporcionar servicios de comunicación a hosts y redes vecinas. [20]

En este capítulo se presenta la arquitectura, funcionamiento e instalación de un clúster de controladores ONOS.

5.1 Arquitectura

A diferencia de OpenDaylight, ONOS está orientado a controlar las redes de proveedores de servicios; pero no quiere decir que no pueda ser utilizado en redes de centros de datos. Considerando esto, la arquitectura de ONOS fue diseñada para cumplir con los siguientes requisitos:

- Alta disponibilidad, escalabilidad y prestaciones
- Fuerte abstracción y simplicidad
- Independencia en el comportamiento de protocolos y servicios
- Separación de tareas y modularidad

La estructura básica de un controlador ONOS está dividida en capas como se muestra en la figura 5 y se detallan a continuación.

- **Core o Núcleo:** Se encarga de presentar una vista lógicamente centralizada del estado de la red y brindar un acceso lógicamente centralizado a las funciones de control de la red.
- **API de Interfaz Norte:** Esta capa se encarga de exponer abstracciones de la capa core a las aplicaciones y servicios de red. Estas abstracciones incluyen desde información de la red (enlaces, hosts, topologías de la red) hasta abstracciones para afectar el estado de la red (programación de flujos).
- **Aplicaciones:** Con los datos que recibe de la interfaz norte, esta capa puede activar diferentes eventos como la generación de reglas de flujo apropiadas que envían a los elementos de la red a través de las siguientes capas. [18]
- **API de Interfaz Sur:** Es una capa de alto nivel que permite la comunicación de la capa core con los elementos de red, haciendo uso de adaptadores (Proveedores) y protocolos específicos.
- **Proveedores o Adaptadores:** Esta capa cumple la función de traductor. Cuando recibe eventos de las capas inferiores y los encapsula para que luego puedan ser procesado o almacenados por las capas superiores. Por otro lado, cuando recibe información de una capa superior se encarga de traducir las abstracciones de alto nivel a protocolos específicos.

- **Protocolos:** Se encarga de la comunicación con los elementos de red, implementando drivers para cada protocolo de comunicación del dispositivo de red.

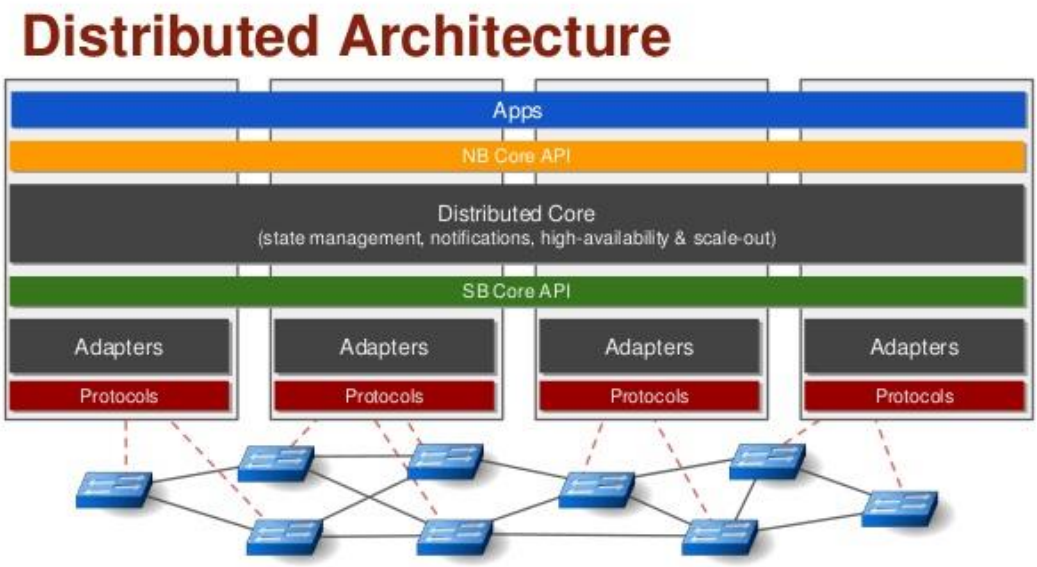

Figura 5. Arquitectura distribuida de ONOS. [20]

5.2 Versiones

Desde el lanzamiento de su primera versión en diciembre de 2014, ONOS ha lanzado un total de 10 versiones. Cada versión recibe el nombre de un ave, en la tabla 3 se citan cada una de ellas y su principal propósito. [21]

Tabla 3. Versiones de ONOS y sus principales objetivos

Versión	**Fecha de Lanzamiento**	**Objetivo**
Avocet	Diciembre 2014	Crear una arquitectura modular que permitiera alta disponibilidad, buenas prestaciones y escalabilidad; para brindar SDN a proveedores de servicio

		y redes de misión crítica.
Blackbird	Marzo 2015	Mejorar las prestaciones y escalabilidad, especialmente en cuanto a operaciones de flujo, cambios de topología, intento de reparación reactiva.
Cardinal	Junio 2015	Agregar nuevos casos de usos, características core, soporte a redes IP-Óptica
Drake	Septiembre 2015	Implementar módulos para seguridad como el de solicitud de usuario y contraseña para ingresar a la interfaz gráfica y línea de comando y soporte de TLS para la comunicación entre nodos. Mejorar la infraestructura implementando subsistemas como el de recolección de métricas.
Emu	Diciembre 2015	Implementar nuevas características de CORD.
Falcon	Marzo 2016	Permitir agregar y eliminar nodos del clúster de manera dinámica.
Goldeneye	Junio 2016	Mejorar las prestaciones y escalabilidad de la interfaz gráfica de usuario
Hummingbird	Septiembre 2016	Implementar RESTCONF cliente y servidor
Ibis	Diciembre 2016	Soportar IGMPv2 y LISP en la interfaz
Junco	Febrero 2017	Mejorar la alta disponibilidad
Kingfisher	Junio 2017	Agregar el modo paquetes/segundo al monitoreo de tráfico en la interfaz web. Soportar dispositivos OpenROADM
Loon	Septiembre 2017	Soportar en su totalidad YANG 1.0
Magpie	Diciembre 2017	Mejorar la detección de falla para una elección de líder más rápida
Nightingale	Mayo 2018	Sin notas de lanzamiento
Owl	Septiembre 2018	Sin notas de lanzamiento
Peacock	Noviembre 2018	Sin notas de lanzamiento

5.3 Funcionamiento del Clúster

Todo clúster se compone de nodos, cada uno con un identificador único. En ONOS lo nodos manejan tres estados que son:

- None(ninguno): Un nodo en este estado no puede interactuar con el elemento de red.
- Master (Maestro): En este estado el nodo tiene control completo del elemento de red, puede leer su estado y realizar en él operaciones de escritura.
- Standby(En espera): Es el estado en el que el nodo tiene conexión con el elemento de red, pero sólo puede realizar operaciones de lectura.

Cuando se inician los nodos todos se encuentran en estado "None", luego el primer nodo que detecta un dispositivo sin máster y que tenga conexión con él, se convertirá en el máster. El clúster cuenta con un líder del subsistema que es el encargado de garantizar que en todo momento cada dispositivo cuente con sólo un controlador maestro, el resto de los controladores se encontrarán en estado de espera. Los controladores que se encuentran en espera están ordenados por prioridad de modo que ante una falla del controlador maestro se pueda tomar rápidamente el control del dispositivo. [20]

A partir de la versión 1.14 el funcionamiento del clúster cambia y hace mayor uso de Atomix que es un sistema distribuido tolerante a fallas. En las versiones anteriores de ONOS cada controlador se tiene embebido un nodo Atomix que es utilizado para formar clúster RAFT, replicar estado y coordinar cambios de estado. Es necesario configurar antes un clúster de Atomix y luego indicarle a ONOS la lista de nodos Atomix a los cuales se debe conectar; los nodos Atomix se encargan de la coordinación y almacenamiento de datos.

5.4 Instalación y Formación de Clúster

ONOS está basado en una plataforma Java por lo que el primer paso será contar con Java instalado, la versión recomendada es la 1.8. Con los siguientes comandos se puede instalar en un equipo con sistema operativo Ubuntu/Debian

```
sudo apt-get install software-properties-common -y && \
sudo add-apt-repository ppa:webupd8team/java -y && \
sudo apt-get update && \
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1
select true" | sudo debconf-set-selections && \
sudo apt-get install oracle-java8-installer oracle-java8-set-default
-y
```

Además de Java, ONOS también requiere que se instale Curl, para ello utilizamos el siguiente comando:

```
sudo apt-get install curl
```

Por defecto ONOS requiere ser instalado en una carpeta "/opt", por lo que debemos ingresar los siguientes comandos para crear la carpeta y ubicarnos en ella.

```
sudo mkdir /opt
cd /opt
```

Una vez nos encontramos en la carpeta "/opt" procedemos descargar ONOS, descomprimirlo y renombrar el directorio creado; para ello se debe ejecutar los siguientes comandos

```
sudo wget -c
http://downloads.onosproject.org/release/onos-1.11.3.tar.gz
sudo tar xzf onos-1.11.3.tar.gz
sudo mv onos-1.11.3 onos
```

Después de instalado ONOS, iniciamos el servicio y observaremos una pantalla como la que se muestra en la Figura 6.

```
/opt/onos/bin/onos-service start
```

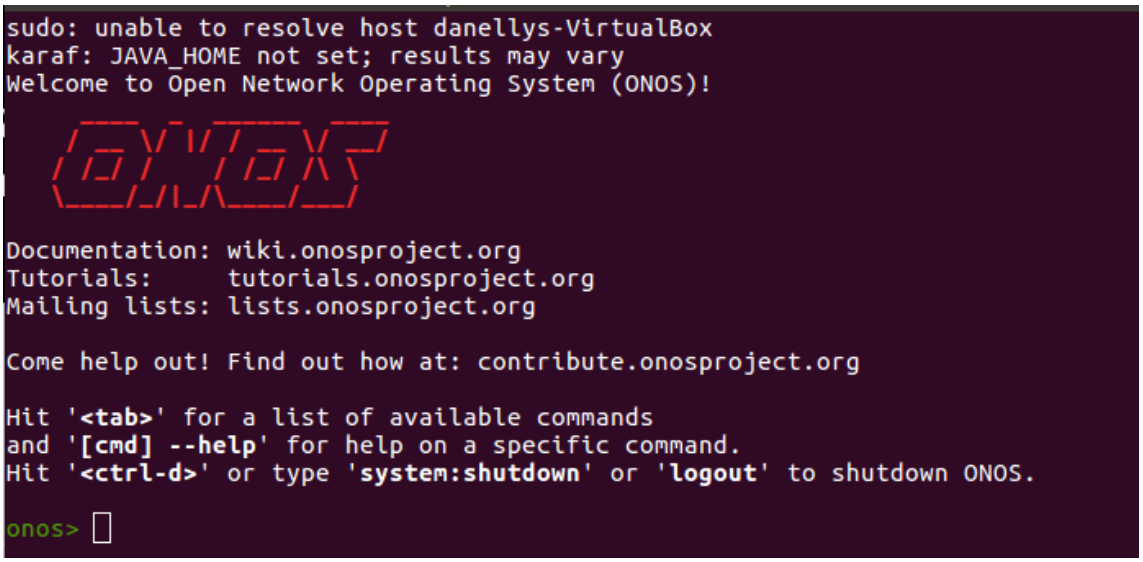

Figura 6. Consola de ONOS

Para realizar la configuración del clúster de ONOS es necesario que todos los nodos que formarán parte de clúster tengan el servicio de ONOS iniciado. Una vez comprobado esto, se debe ejecutar desde uno de los nodos el script para formar cluster que se encuentra en la carpeta "bin", se debe colocar las direcciones IP de cada uno de los nodos que formarán el clúster. A continuación, se muestra un ejemplo para un cluster de 3 controladores.

```
/opt/onos/bin/onos-form-cluster 10.10.1.1 10.10.1.2 10.10.1.3
```

Para verificar que el clúster se ha formado, se puede escribir desde la interfaz de karaf el comando "nodes" y se podrán observar las direcciones IP de todos los nodos que conforman el clúster.

Para completar la instalación es necesario activar las aplicaciones para permitir que se comunique con dispositivos que manejan openflow, que se realice balanceo de carga y se pueda proporcionar funcionalidades de conectividad. Esto se realiza ejecutando los siguientes comandos en la línea de comandos de ONOS:

```
app activate org.onosproject.openflow

app activate org.onosproject.mlb

app activate org.onosproject.fwd
```

Al igual que con OpenDaylight, los conmutadores de Open vSwitch deben conectarse a todos los nodos del clúster para brindar alta disponibilidad.

5.5 Seguridad en ONOS

En el análisis de seguridad y prestaciones de ONOS [22] realizan recomendaciones que debemos tomar en cuanto a seguridad, todas ellas relacionadas a los valores por defecto incorporados en ONOS.

- Contraseñas por defecto: El usuario "onos" junto con la contraseña "karaf" son las credenciales por defecto en ONOS y brindan acceso a la interfaz web, la interfaz de línea de comandos y REST API.
- Interfaz Norte: Por defecto acceso a la interfaz norte se brinda por medio de HTTP. A pesar de que por defecto la autenticación está habilitada, si no se utiliza HTTPS no hay seguridad en la transferencia de datos entre el cliente y el servidor ONOS. No se recomienda utilizar certificados autofirmados ya que puede presentar problemas con ciertos navegadores y REST API. En la web de ONOS se explica cómo habilitar HTTPS de manera manual.
- Interfaz Sur: Por defecto la interfaz sur no está encriptada ni se autentican los dispositivos conectados. Se recomienda habilitar SSL para encriptar los datos para prevenir la observación y manipulación de datos. También es recomendable habilitar TLS para que realice autenticación entre el servidor y los dispositivos de red una vez han sido conectados.
- Interfaz Este/Oeste: El protocolo TLS no viene configurado por defecto para la comunicación entre controladores y es altamente recomendado habilitarlo. Los detalles sobre cómo habilitar TLS para la comunicación entre controladores se puede encontrar en la wiki de ONOS. [20]

5.6 Módulos Utilizados en ONOS

En este apartado se describen los módulos empleados en ONOS para el funcionamiento de la herramienta.

- Org.onosproject.mlb: Esta aplicación permite que los controladores que forman un clúster realicen automáticamente un balanceo de la cantidad de conmutadores conectados. Por ejemplo, si se tiene un clúster de 3 controladores y se detectan que existen 12 conmutadores, entonces se realizará un balanceo automático de modo que cada controlador sea el máster de 4 controladores; si uno de los controladores falla automáticamente se producirá un rebalanceo y cada controlador quedará siendo el máster de 6 conmutadores.
- Org.onosproject.fwd: Esta aplicación permite el reenvío de paquetes tanto en capa 2 como en capa 3, también maneja direcciones IPv6. No es necesario instalar ningún módulo STP para eliminar bucles en la red.
- Org.onosproject.openflow: Permite la comunicación con dispositivos que manejan el protocolo openflow.

5.7 Incidencias en el Uso de ONOS

Entre las incidencias presentadas con el uso de ONOS está que se ha tenido que trabajar con la versión de ONOS 1.11.3 ya que en versiones superiores se producían errores al momento de formar el clúster, esto está asociado a un bug con el uso de contenedores.

Otra de las incidencias presentadas está en que si la aplicación para el reenvío de paquetes estaba activada y luego se inician los controladores también se presentan errores, para esto se decidió dejar desactivada esta aplicación (org.onosproject.fwd) y que sea activada manualmente por el usuario al momento de realizar pruebas de conectividad.

6 Diseño de la Herramienta

En este capítulo se exponen los requisitos funcionales y de instalación tomados en cuenta al momento de diseñar la herramienta y la manera en la cada una de estas funcionalidades fueron implementadas.

Esta herramienta es desarrollada con una arquitectura plana de controladores, ya que es la arquitectura utilizada por ONOS y OpenDaylight que son los controladores que se

utilizarán en la herramienta. En cuanto a los conmutadores, la herramienta permite seleccionar entre una topología en malla completa o anillo.

6.1 Requisitos Funcionales

Para el diseño de la herramienta se establecieron los siguientes requisitos funcionales:

6.1.1 Interfaz Gráfica

La herramienta debe contar con una interfaz gráfica que permita al usuario modificar los parámetros de entrada conforme a sus necesidades.

6.1.2 Creación del Clúster de Controladores

La herramienta desarrollada debe permitir la creación de un clúster de controladores con alta disponibilidad utilizando OpenDaylight u ONOS, para esto el usuario seleccionará de un menú desplegable el tipo y cantidad de controladores que desea formen el clúster.

Se establece que la cantidad mínima de controladores para formar un clúster debe ser 3, conforme a lo indicado por OpenDaylight y ONOS para un clúster de alta disponibilidad. La herramienta también permitirá desplegar entornos con 1 controlador, para analizar así la diferencia con los entornos de múltiples controladores.

6.1.3 Creación de Topología

La herramienta debe permitir la creación de un clúster de controladores usando VNX. La red manejada por el clúster de controladores se puede crear dentro del escenario VNX, o utilizando comandos Open vSwitch que van añadiendo conmutadores a la red. El primer enfoque es razonable cuando el número de conmutadores no es muy grande y el escenario VNX resultante es manejable, pero en caso de números elevados de conmutadores (mayores a 100) es preferible el segundo. Para ciertos experimentos en los que se requiere tener el clúster estable antes de agregar los conmutadores será necesario agregar la red a manejar por medio de comandos Open vSwitch.

6.1.4 Pruebas de Prestaciones

La herramienta debe permitir al usuario realizar un análisis del entorno desplegado en cuanto a escalabilidad, disponibilidad y rendimiento.

6.2 Herramientas y Lenguajes Utilizados

En el desarrollo de esta herramienta se hizo uso de software de código abierto y lenguajes de programación que se citan a continuación:

- Python: La herramienta fue desarrollada empleando Python. Python es un lenguaje de programación interpretado, orientado a objeto que tiene como objetivo permitir la creación de un código legible y de rápido desarrollo. Entre

sus características está que es de tipado dinámico, es decir que una variable puede tomar valores de distintos tipos en diferentes momentos. [23]

- Virtual Network over Linux (VNX): VNX es una herramienta de código abierto para virtualización de propósito general, permite desplegar escenarios de red virtualizados constituidos por máquinas virtuales, toma como entrada un archivo XML que por medio de un lenguaje de especificación VNX permite describir la topología de la red. [24]
- TShark: Es una versión de wireshark orientada a terminal que permite capturar y mostrar paquetes. [25]
- OpenDaylight: Uno de los controladores empleados en esta herramienta fue OpenDaylight, se utilizó la penúltima versión (Nitrogen) que al momento del desarrollo era la última versión estable. Los detalles de este controlador se encuentran en el capítulo 4. [15]
- ONOS: Otro de los controladores a utilizar es ONOS, en su versión 1.11.8. Las características de este controlador fueron definidas en el capítulo anterior. [20]
- Integration Tool Test: Conjunto de programas escritos en lenguaje de programación Python que fueron desarrollados para validación de la implementación y prestaciones en redes definidas por software que emplean el controlador OpenDaylight. [26]

6.3 Requisitos de Implementación

La herramienta está diseñada para ser utilizada en equipos con sistemas operativos Ubuntu y requiere que cuente con los siguientes elementos instalados:

- Python: Se debe tener python instalado en el equipo sobre el cual se ejecutará la herramienta. Además, debe tener instaladas las librerías de python "tkinter" utilizada para el despliegue de la interfaz gráficas, "netaddr" empleada en el manejo de las direcciones IP y "requests" que permite el uso de una aplicación para enviar flujos a los controladores.
- VNX: Es necesario tener instalado VNX para el despliegue virtualizado de la infraestructura de controladores de la red. En [24] se pueden encontrar los pasos para su instalación.
- Sistema de Ficheros: Se ha desarrollado un sistema de ficheros que cuenta con OpenDaylight, ONOS, Java y herramientas para la medición de prestaciones instaladas. Este sistema de ficheros es utilizado por VNX para el despliegue de los contenedores LXC.
- Permiso de Superusuario: Para utilizar la herramienta se debe contar con permiso de superusuario ya que es necesario para la ejecución de VNX.

- Tshark: La máquina sobre la cual se ejecuten las pruebas debe tener instalado Tshark para la captura y análisis de paquetes de red.

6.4 Módulos de la Herramienta

Para el desarrollo de la herramienta se decidió utilizar el paradigma de programación modular. La programación modular se basa en la división de un programa en subprogramas también llamados módulos, lo que permite pasar de un programa complejo a varios programas simples y hace que el código sea más manejable y legible.

En la figura 7 se muestran los módulos y submódulos que forman la herramienta desarrollada y a continuación se explica cada uno de ellos.

Figura 7. Diagrama de los módulos de la herramienta

6.4.1 Interfaz Gráfica de Usuarios

Uno de los módulos del programa se encarga de crear una interfaz gráfica de usuario, permitiendo al usuario ingresar los datos necesarios para el despliegue de la herramienta de una manera intuitivo. Se encarga también de enviar datos recibidos a otros módulos para su posterior procesado.

A fin de crear una interfaz gráfica se utiliza el paquete estándar de Python, Tkinter [27]. Entre las funcionalidades de Tkinter empleadas en la herramienta están:

- Etiquetas (Label): Es utilizada para mostrar texto o imágenes.
- Cuadro de Texto (Entry): Permite al usuario introducir los parámetros de entrada a la herramienta.
- Botones (Button): Contienen una función asociada que es ejecutada al momento de presionar el botón.
- Caja de Mensajes (MessageBox): En la aplicación desarrollada se utilizan cajas de mensajes de error cuando en uno de los campos se coloca un valor no válido.

- Menú de Opciones (OptionMenu): Permite al usuario seleccionar entre las opciones desplegadas.

Para empezar a desarrollar la interfaz gráfica se debe importar los paquetes de Tkinter necesarios y crear una ventana sobre la cual se irán agregando etiquetas, botones, cuadros de texto, etc.

```
from tkinter import *
from tkinter import font
from tkinter import messagebox

root = Tk() #Creando ventana para interfaz gráfica
```

Para organizar los elementos, la ventana se maneja como una cuadrícula (Grid) y a cada elemento se le asigna unas coordenadas que determinan su ubicación en dicha cuadrícula como se muestra en la figura.

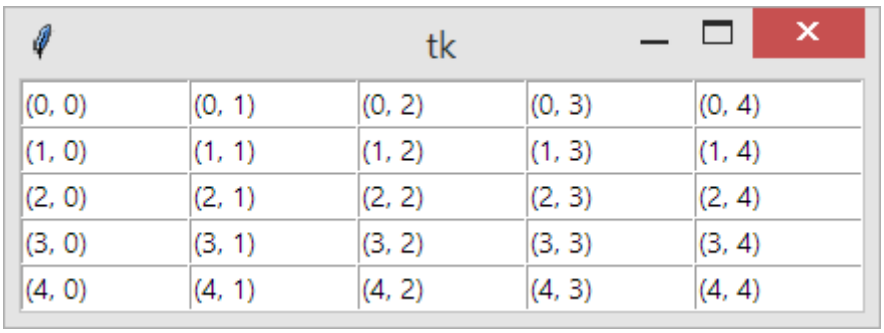

Figura 8. Cuadrícula de TkInter

La función grid tiene diferentes propiedades que se pueden agregar a los elementos para mejorar su ubicación, entre los utilizados para el desarrollo de esta interfaz están:

- Columnspan: Permite unir varias columnas y colocar allí un elemento.
- Pady/Paxy: Indica cuántos píxeles colocar en la parte exterior de ese elemento vertical (Pady) u horizontalmente (Padx). El valor por defecto es 0.
- Sticky: Sirve para indicar desde dónde se expande elemento. La nomenclatura por utilizar es N (Norte), S (Sur), E(Este), W (Oeste). Por ejemplo, si colocamos sticky=E indicamos que ese elemento se encuentra alineado hacia la derecha (este). El valor por defecto es en el centro.

Un ejemplo de cómo utilizar las propiedades antes mencionadas se muestra a continuación:

```
Button1.grid(row=3, column=1, pady=(10,30), padx=(5,5),
sticky=W+E)
```

La figura 9 muestra la interfaz gráfica de usuario para la aplicación diseñada. Como se observa consta de dos partes una orientada al despliegue del escenario y otra para la realización de los experimentos.

Figura 9. Interfaz gráfica de la herramienta

6.4.2 Manejo de Escenarios

El módulo para manejo de escenarios recibe parámetros ingresados por el usuario a través de la interfaz gráfica. Cuenta con los siguientes submódulos que trabajan en conjunto para crear el escenario:

- **Validación de Datos de Topología:** Este submódulo se encarga de verificar que los datos ingresados por el usuario y que posteriormente son utilizados para desplegar el escenario sean del tipo adecuado, es decir, que en el cuadro de texto para dirección IP se ingrese una dirección IP, que en un campo numérico se ingrese una variable numérica y no una letra.

Los datos que verifica este módulo son: la cantidad de conmutador, la cantidad de host y la dirección IP. En caso de que el valor ingresado no sea válido, este módulo se encarga de mostrar un mensaje de error como en la figura 10 y no permite continuar con la ejecución del programa.

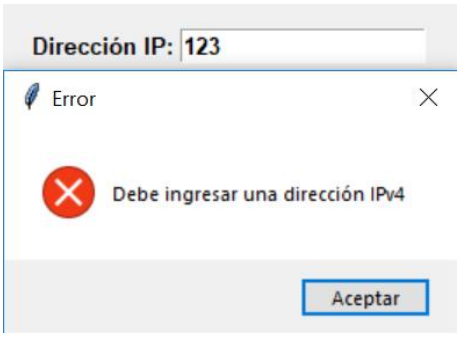

Figura 10. Mensaje de error al ingresar un valor no válido

- **Creación de XML:** Este submódulo es el encargado de crear un archivo XML que contenga todos los elementos indicados por el usuario, para ello se divide en subprogramas cada uno encargado de crear una parte del XML correspondiente a ciertos elementos de la red. Los archivos XML ejecutados por la herramienta deben ser creados con este módulo, no se permite la ejecución de XML externos a la herramienta. El archivo XML se generará en la misma carpeta de la herramienta, puede ser copiado para su utilización fuera de la herramienta, sin embargo, una vez se detenga la herramienta este archivo se eliminará.

  El subprograma encargado de la parte del XML del conmutador toma como datos de entrada la cantidad de conmutadores, la cantidad de controladores, el tipo de topología (malla o anillo) y la dirección IP especificada por el usuario. Maneja distintos ciclos "For", uno de ellos encargado de crear la cantidad de conmutadores indicados por el cliente, otro de enlazar las direcciones IP de los controladores para que los conmutadores se puedan conectar a ellos y otro ciclo para las conexiones entre conmutadores. El XML para 2 conmutadores, en un entorno de 3 controladores quedaría como se muestra a continuación:

```
<net name="Net1" mode="openvswitch" hwaddr="00:00:00:00:00:01"
controller="tcp:10.1.0.5:6653 tcp:10.1.0.6:6653 tcp:10.1.0.7:6653"
fail_mode='secure' of_version="OpenFlow13">
<connection name='link12' net='Net2' >
</connection>
</net>
<net name="Net2" mode="openvswitch" hwaddr="00:00:00:00:00:02"
controller="tcp:10.1.0.5:6653 tcp:10.1.0.6:6653 tcp:10.1.0.7:6653"
fail_mode='secure' of_version="OpenFlow13">
<connection name='link21' net='Net1' >
</connection>
</net>
```

El subprograma encargado de los XML correspondiente al Host toma como datos de entrada la cantidad de conmutadores y de hosts; cuenta con dos ciclos que permiten crear hosts con sistema operativo Ubuntu 16.04 y luego enlazarlos a su conmutador correspondiente. Cada conmutador estará en una subred diferente, los host conectados al conmutador Net1 estarán en la subred 192.168.1.0724, los conectados al conmutador Net2 estarán en la subred 192.168.2.0/24 y así sucesivamente. Si el usuario define que desea dos conmutadores, con dos hosts por conmutador, el subprograma crearía un XML como el que se muestra a continuación, donde los dos primeros host están conectados al conmutador Net1 y los dos siguientes al conmutador de Net2:

```
<vm name = "PC1_1" type="lxc" exec_mode = "lxc-attach" arch = "x86_64">
<filesystem
type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-16.04-
v025</filesystem>
<if id="1" net="Net1">
<ipv4>192.168.1.2/24</ipv4>
</if> </vm><vm name = "PC1_2" type="lxc" exec_mode = "lxc-attach" arch =
"x86_64">
<filesystem
type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-16.04-
v025</filesystem>
<if id="1" net="Net1">
<ipv4>192.168.1.3/24</ipv4>
</if> </vm><vm name = "PC2_1" type="lxc" exec_mode = "lxc-attach" arch =
"x86_64">
<filesystem
type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-16.04-
v025</filesystem>
<if id="1" net="Net2">
<ipv4>192.168.2.2/24</ipv4>
</if> </vm><vm name = "PC2_2" type="lxc" exec_mode = "lxc-attach" arch =
"x86_64">
<filesystem
type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-16.04-
v025</filesystem>
<if id="1" net="Net2">
<ipv4>192.168.2.3/24</ipv4>
</if> </vm>
```

El siguiente subprograma es el encargado de crear la parte de XML de los controladores, los datos que recibe como entrada son el tipo y cantidad de controladores, así como también una dirección IP que se asignará localmente para tener acceso a los controladores, a partir de allí las siguientes direcciones IP serán las correspondiente a los controladores. Este subprograma a su vez se subdivide y dependiendo del tipo de controlador seleccionado se ejecutará ya sea el de ONOS o de ODL. En ambos casos se cuenta con un ciclo que crea la

cantidad controladores indicados por el usuario, estos controladores tendrán sistema operativo Ubuntu 16.04.

En el caso de ODL como se muestra a continuación en la secuencia de arranque "boot" se ejecuta el script que permite la creación del clúster.

```
<vm name = "ODL1" type="lxc" exec_mode = "lxc-attach" arch = "x86_64">
<filesystem type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-16.04-
v025</filesystem>
<if id="1" net="Net0">
<ipv4>10.1.0.5/24</ipv4>
</if>
<exec seq="on_boot" type="verbatim"> /sdn/karaf-0.7.2/bin/configure_cluster.sh 1 10.1.0.5
10.1.0.6 10.1.0.7 </exec>
</vm>
```

El submódulo de topología XML define la subred a la cual se conectan los controladores:

```
<host>
<hostif net="Net0">
<ipv4>10.1.0.1/24</ipv4>
 </hostif>
 </host>
</vnx>
```

- **Comandos VNX:** El submódulo de programa VNX se encarga de ejecutar los comandos VNX que lanzan el escenario, muestran la topología de la red desplegada o detienen el escenario.

  Una vez el programa crea el archivo xml con la estructura de red cuyo nombre es escenario.xml, se utiliza el siguiente comando de vnx para desplegar el escenario:

  ```
  sudo vnx -f escenario.xml -v --create
  ```

  Luego de creado el escenario se utilizan unos scripts que contienen los comandos "lxc-attach -n" para dejar en ejecución los clústeres de controladores.

  La función de mostrar la topología del escenario es realizada por medio del comando de vnx:

  ```
  sudo vnx -f escenario.xml -v --show-map
  ```

  Por otro lado, para detener el escenario el programa de Python ejecuta:

  ```
  sudo vnx -f escenario.xml -v --P"
  ```

6.4.3 Pruebas de Entornos Desplegados

Una vez creado el escenario, el módulo de pruebas de entornos desplegados permite la ejecución de pruebas para evaluar la escalabilidad, tolerancia a falla y prestaciones de los entornos. Este módulo se comunica con la interfaz gráfica de usuario por medio de la cual se indican el tipo de pruebas que se desea realizar; y se compone de los siguientes submódulos:

- **Validación de Datos de Tráfico:** Este submódulo se encarga de verificar que los datos ingresados por el usuario y que se utilizan para las pruebas de prestaciones sean del tipo adecuado. La cantidad de solicitudes y los flujos deben ser números enteros, de lo contrario se enviará un mensaje de error y no se podrá seguir con la ejecución de las pruebas.
- **Ejecución de Pruebas:** Una vez se comprueba que los datos ingresados por el usuario para las pruebas son del tipo adecuado, este submódulo se encarga de poner en ejecución las distintas pruebas. Aquí se agrupan las diferentes funciones que permiten la ejecución de los experimentos descritos en la sección 6.5.

  La función de "Capturar Paquetes" utiliza un script tshark.sh que despliega un terminal en el que se capturan los paquetes TCP SYN enviados desde los conmutadores Open vSwitch hacia los controlados y los paquetes filtrados son almacenados en un archivo llamado OVSEvents. El comando que se ejecuta es el siguiente:

  La función "Calcular Latencia" obtiene los logs de Karaf ya sea de ONOS u OpenDaylight y realiza un filtrado entre los logs de todos los controladores para encontrar el último evento de topología. Se extrae la parte de los logs correspondiente a la fecha y se utiliza la librería "date" de python para darle formato. Finalmente se obtiene la latencia por medio de diferencia de la fecha de los logs de Karaf con los logs OVSEvents que fueron generados al capturar los paquetes en tshark.

6.5 Diseño de Experimentos

A continuación, se describen los experimentos propuestos para la evaluación de entornos de múltiples controladores. Estas pruebas se basan en tres factores a analizar: escalabilidad, tolerancia a falla y rendimiento.

6.5.1 Tiempo de Descubrimiento de Topología

A fin de verificar la escalabilidad en los entornos desplegados se realiza uno de los experimentos propuestos por ONOS [28] que consiste en medir cuánto tiempo les toma a todos los controladores de un clúster descubrir una determinada topología,

siendo esta latencia la diferencia entre el tiempo en el que se da el último evento de cambio de topología y el de momento en el que se envía el primer mensaje de TCP SYN desde el conmutador.

Para la ejecución de este experimento se requiere primero tener un clúster estable, luego crear los conmutadores y finalmente conectarlos a los controladores. Es por ello por lo que se debe utilizar VNX sólo para desplegar la parte del entorno correspondiente a los controladores, y agregar los conmutadores por medio del botón llamado "Agregar Conmutadores OVS".

Para la realización de este experimento se creó un botón en la interfaz que permite lanzar tshark para capturar paquetes de la red y uno que calcula la latencia haciendo uso las capturas de tshark y de los logs de la consola Karaf. En tshark se filtran los paquetes TCP SYN que inician la comunicación entre los conmutadores Open vSwitch y los controladores. En los logs de karaf se filtran los eventos "TopologyManager" en el caso de ONOS y "DeviceManagerImpl" para OpenDaylight. Estos eventos registran los cambios en la topología. La latencia medida es la diferencia entre el primer paquete de SYN enviado desde uno de los conmutadores hasta el último evento de cambio de topología registrado en los controladores.

Para la ejecución de este experimento se debe primero presionar el botón "Capturar Paquetes" que lanzará una consola con tshark, posteriormente indicar la cantidad de conmutadores a agregar y presionar el botón agregar conmutadores. Una vez se verifica que todos los conmutadores se han agregado a los controladores, se debe presionar el botón "Calcular Latencia" y se mostrará el tiempo total en segundos. Luego se debe presionar el botón "Eliminar Conmutadores" y repetir este procedimiento para otra cantidad de conmutadores.

En esta prueba además de contabilizar la latencia al ir aumentando la cantidad de conmutadores conectados a la red, se verifica que la información almacenada en los controladores sea estable y correcta. Se realizan pruebas de ir agregando conmutadores hasta encontrar la cantidad máxima soportada por el escenario.

Se realizaron pruebas de escalabilidad utilizando el controlador ONOS y OpenDaylight, en cada caso se desplegaron 4 tipos de escenarios: un escenario con un solo nodo (sin clúster) y otros con clúster de 3, 4 y 5 nodos. El propósito de la herramienta es evaluar escenarios de múltiples controladores, sin embargo, se realiza la prueba de escalabilidad con un solo nodo a fin de estudiar la diferencia con los escenarios de múltiples controladores.

6.5.2 Latencia en Eventos de Conmutadores

Con este experimento se busca verificar tanto escalabilidad como la reacción ante nuevos eventos o fallas.

Para esta prueba se agrega y se elimina un conmutador. El conmutador estará conectado a un solo controlador y se mide el tiempo en el que todos los nodos miembros del clúster han registrado el cambio de la topología. Para el caso de agregar un nuevo conmutador al igual que para la prueba anterior la latencia está dada por la diferencia de tiempo entre el momento en que se envía el SYN/ACK desde el conmutador Open vSwitch hasta cuando se da el último evento de cambio de topología en los controladores. Para el evento de eliminar conmutador se mide la diferencia de tiempo entre el momento en que se de baja el conmutador en Open vSwitch hasta el último cambio de topología en los controladores. Es importante mencionar que en esta prueba no se tiene en cuenta el tiempo de conectarlo a la red de conmutadores.

Para realizar estas pruebas se han creado dos botones en la interfaz gráfica, cada uno simula uno de los eventos indicados, calcula la latencia haciendo uso de capturas de tshark y los logs de karaf, posteriormente muestra el resultado en la consola.

6.5.3 Tolerancia a falla

Esta prueba no automatizada consiste en lanzar una red con una cantidad determinada de controladores, se propone una de 5 conmutadores y 5 hosts por conmutador. Luego ir dando de baja los controladores, según la máxima tolerancia soportada y verificar que el resto de los conmutadores continúen mostrando la red sin inconsistencias. Finalmente iniciar nuevamente el controlador que se había dado de baja y verificar que muestra la topología de la red correctamente. Repetir este procedimiento para cada uno de los miembros del clúster.

6.5.4 Instalación de Flujos en los Conmutadores

Para este experimento se hace uso de una serie de scripts desarrollados en python por OpenDaylight bajo el proyecto Cross. Se escogió utilizar esta herramienta ya que cuenta también con un script para realizar las pruebas en ONOS.

Los scripts por utilizar funcionan enviando una serie de flujos a los controladores desde la interfaz norte y mide el tiempo en el que tardan estos flujos en instalarse en los conmutadores que se encuentren conectados. Los parámetros de entrada que toma son: dirección IP del controlador a la cual se enviaran los flujos, la cantidad de flujos a instalar en cada conmutador y cuántos flujos se desea que maneje cada solicitud. Este script lo primero que realiza es verificar cuantos conmutadores se encuentran en la red y los flujos ya existentes- Luego entre los conmutadores detectados una realiza una selección aleatoria para decidir sobre qué conmutador se instala cada flujo; posteriormente va

enviando estos flujos al controlador indicado por medio de solicitudes REST. Una vez se han enviado todos los flujos empieza a consultar el controlador cada segundo para confirmar si todos los flujos han sido instalados en los conmutadores. Para obtener mejor precisión en los resultados se modificaron los scripts para que la consulta a los controladores se realice cada 0.25 segundos. Se modificó en el script que los flujos no sean permanentes, si no que tengan una duración de 60 segundos, de modo que los resultados de una prueba no se vean afectados por flujos ingresados anteriormente.

En el experimento se propone realizar pruebas manteniendo una misma cantidad de conmutadores para todos los escenarios, ir variando la cantidad de flujos y contabilizar el tiempo que tarda en instalarse los flujos en los conmutadores. Este procedimiento se debe repetir al menos dos veces para cada escenario, cambiando la dirección IP del controlador hacia la cual se envían las solicitudes para analizar así la diferencia entre el balanceo ofrecido por ONOS y el hecho de que OpenDaylight no ofrezca esta característica.

6.5.5 Máxima escalabilidad

En esta prueba se propone crear una red de conmutadores Open vSwitch utilizando el botón "Agregar conmutadores OVS" y luego ir desplegando diferentes entornos de múltiples controladores para verificar si la cantidad de conmutadores creados es soportada o si se producen desconexiones. Si el entorno soporta la cantidad de conmutadores correctamente sin producir desconexiones entonces eliminar los conmutadores OVS y repetir el procedimiento para validar una red de mayores dimensiones.

6.6 Creación de FileSystem

Para iniciar a crear el clúster de controladores se requiere que VNX sea capaz de desplegar máquinas virtuales con un filesystem que tenga instalado OpenDaylight, ONOS y los módulos necesarios.

Al utilizar VNX es posible descargar diferentes filesystem para su posterior utilización escribiendo el comando "vnx_download_rootfs". El repositorio de VNX cuenta ya con un fichero que contiene OpenDaylight, sin embargo, esta versión es del año 2015, por lo que se decidió descargar un filesystem con el sistema operativo Ubuntu y realizar todas las instalaciones necesarias para ejecutar allí una versión más actualizada de OpenDaylight y ONOS.

Con VNX instalado se procedió a descargar un filesystem de Ubuntu 16.04 64 bits sin interfaz gráfica y luego en la carpeta *"/usr/share/vnx/filesystems"* se ejecutó el comando *"sudo vnx --modify-rootfs vnx_rootfs_lxc_ubuntu64-16.04-v025"* que permite realizar

cambios en el filesystem. Luego de esto se realizó la instalación de OpenDaylight y ONOS siguiendo los pasos descritos en los capítulos 4 y 5 respectivamente.

7 Pruebas de la Herramienta

En este capítulo se describen las pruebas que se han llevado a cabo para validar el correcto funcionamiento de la herramienta desarrollada

7.1 Verificación de Módulo de Validación de Datos

Se realizaron pruebas ingresando valores erróneos en cada uno de los campos disponibles en la interfaz gráfica, validando así que los errores son detectados correctamente por la herramienta.

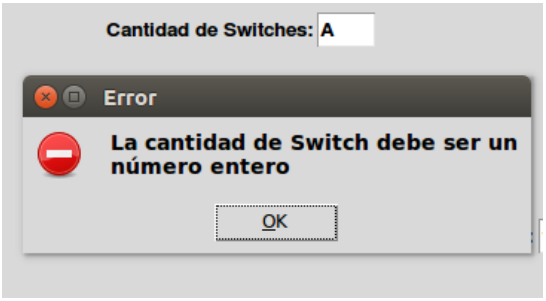

Figura 11. Comprobación de error al ingresar un valor no válido en el campo de cantidad de conmutadores

.

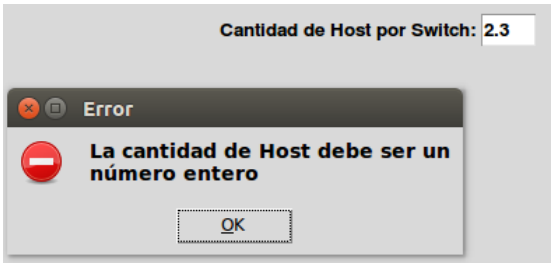

Figura 12. Comprobación de error al ingresar un valor no válido en el campo de cantidad de host por conmutadores

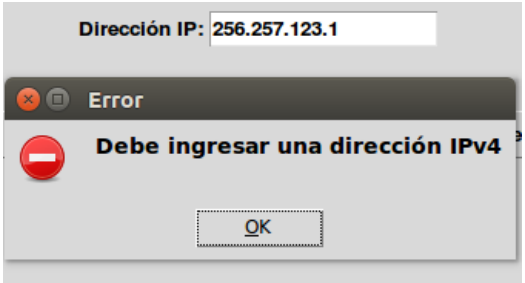

Figura 13. Comprobación de error al ingresar un valor no válido en el campo de dirección IP

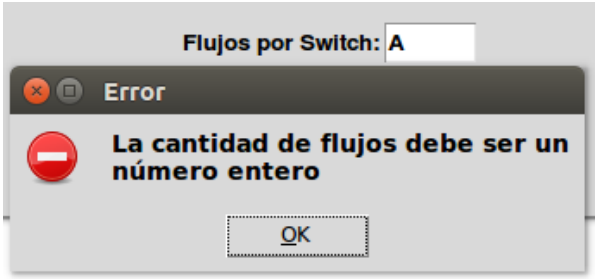

Figura 14. Comprobación de error un valor no válido en el campo de flujos

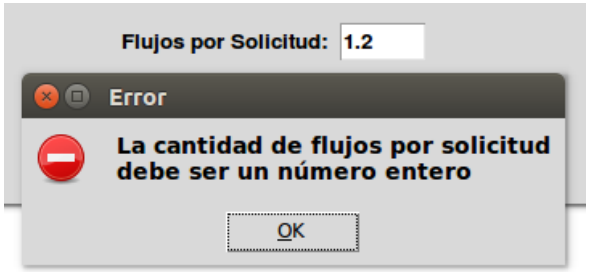

Figura 15. Comprobación de error un valor no válido en el campo de flujos por solicitud.

7.2 Validación de Formación de Clúster

Una vez ingresados datos correctos a la herramienta, se procedió a desplegar el escenario con el fin de validar que el clúster de controladores se forme correctamente.

En el caso de ONOS se indicó un clúster de 5 controladores y se escribió desde la consola de uno de los nodos el comando "nodes", como se muestra en la figura 16 se valida que el clúster se formó correctamente. También se accedió a la interfaz web de uno de los nodos para visualizar los miembros del clúster como se muestra en la figura 17. Para acceder a la interfaz web de uno de los nodos de ONOS se debe escribir en uno de los navegadores web "http://Direccion_IP:8181/onos/ui/login.html" y colocar las credenciales por defecto de ONOS (usuario:onos, contraseña:rocks).

```
onos>
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos> nodes
id=10.10.10.2, address=10.10.10.2:9876, state=ACTIVE, version=1.11.2, updated=10s ago *
id=10.10.10.3, address=10.10.10.3:9876, state=READY, version=1.11.2, updated=3m35s ago
id=10.10.10.4, address=10.10.10.4:9876, state=READY, version=1.11.2, updated=3m49s ago
id=10.10.10.5, address=10.10.10.5:9876, state=READY, version=1.11.2, updated=3m32s ago
id=10.10.10.6, address=10.10.10.6:9876, state=INACTIVE, version=1.11.2, updated=5s ago
onos>
```

Figura 16. Validación de clúster por la consola de ONOS.

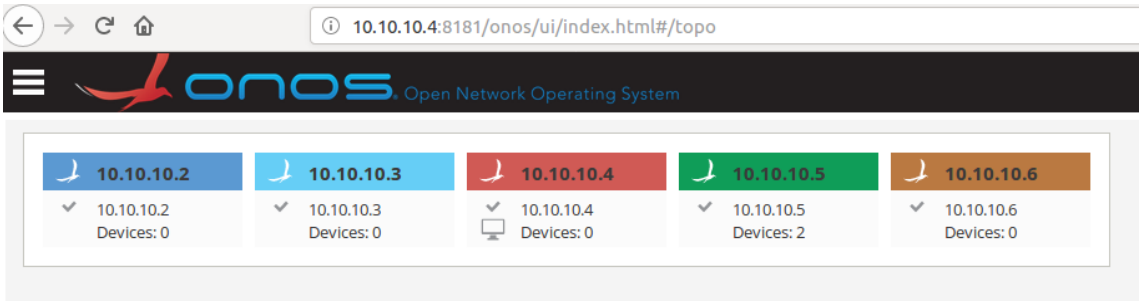

Figura 17. Validación de clúster por la interfaz web de ONOS.

7.3 Validación de Creación de Conmutadores

Se validó que los conmutadores creados corresponden con la cantidad indicada por medio de la interfaz de usuario y que se comunicaran con los controladores. Para esta prueba se desplegó el escenario con 4 conmutadores.

En ONOS se valida por medio de la consola de uno de los nodos escribiendo el comando "devices" o por medio de la interfaz web como se puede observar en la figura 18.

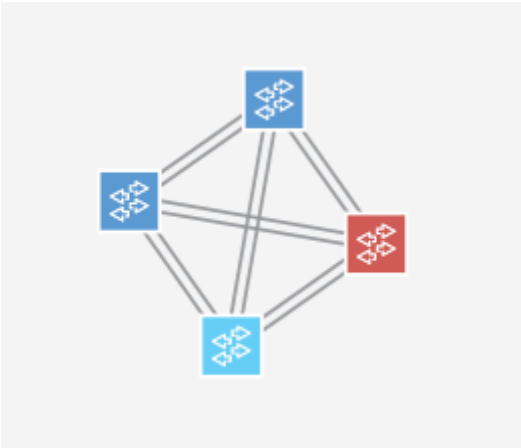

Figura 18. Conmutadores en la interfaz de ONOS. Los diferentes colores de cada uno identifican el nodo maestro

En OpenDaylight también se validó por medio de la interfaz web como se muestra en la figura 19. Para acceder a la interfaz web de OpenDaylight se debe colocar en el navegador web "http://Dirección_IP:8181/index.html"

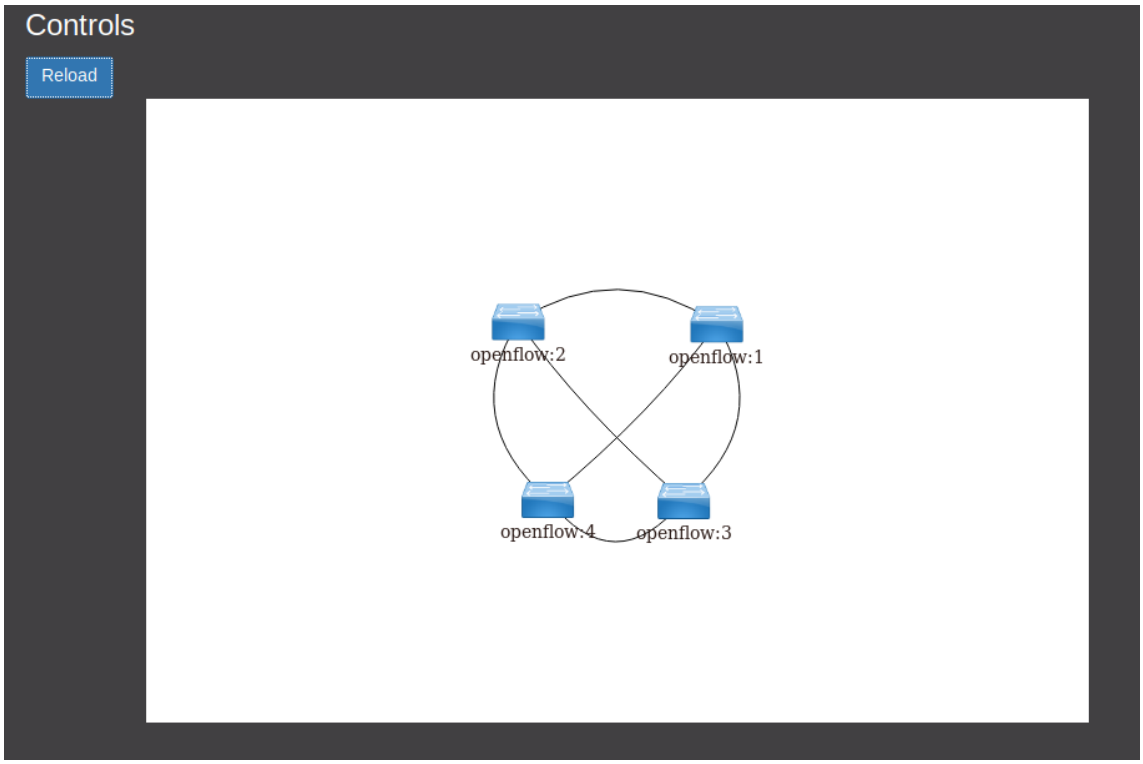

Figura 19. Conmutadores en la interfaz de OpenDaylight.

7.4 Verificación del Botón Mostrar Topología

Se creó una topología de 3 nodos con controladores ONOS, 5 conmutadores conectados en anillo y 3 hosts por conmutador. Al presionar el botón de "Mostrar Topología" se muestra la imagen de la figura 20 que corresponde con la topología creada.

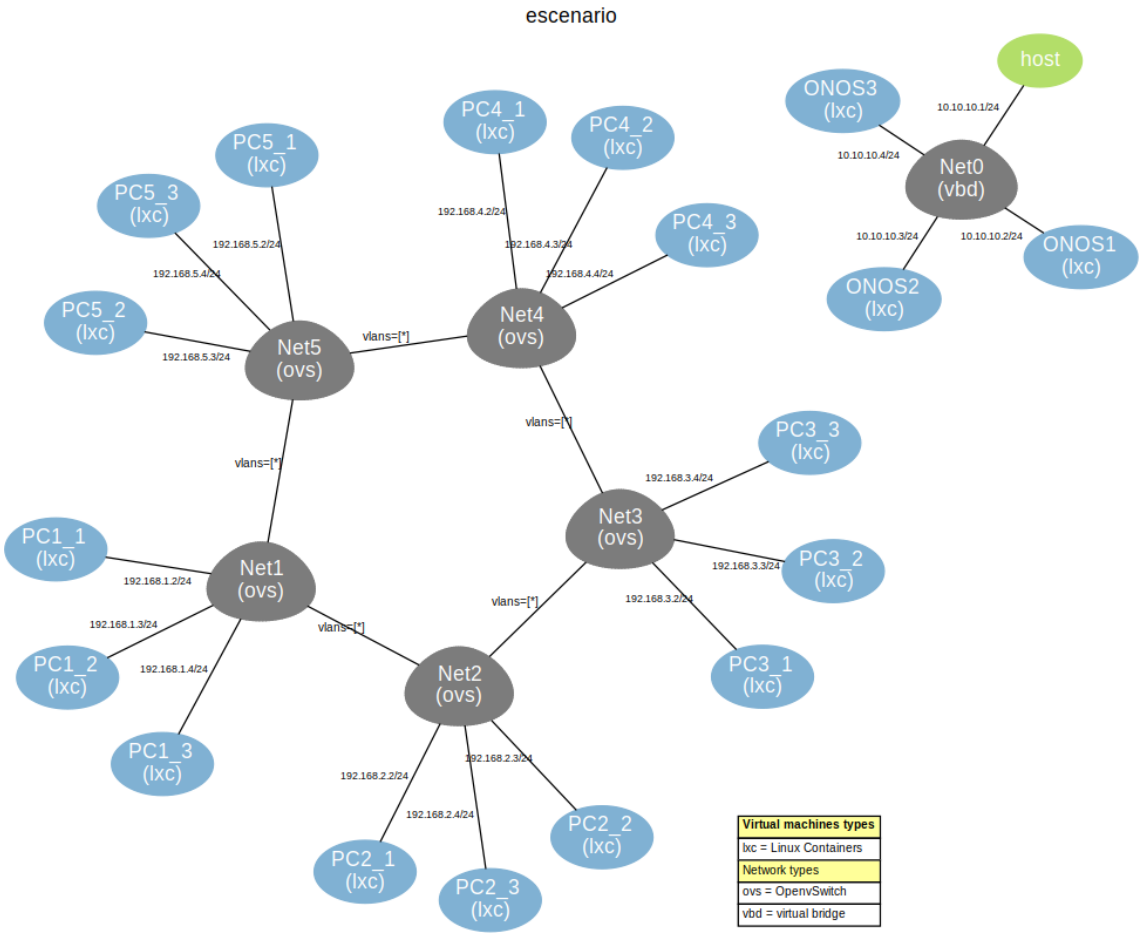

Figura 20. Topología del escenario creado desplegada en VNX.

7.5 Pruebas de Conectividad

Se creó un escenario con un clúster de 3 controladores ONOS, 4 conmutadores conectados en malla y dos hosts por conmutador, posteriormente se realizó prueba ping entre todo ellos. Para OpenDaylight se creó un escenario con 3 controladores, dos conmutadores y 2 host por conmutador.

Las pruebas de ping efectuadas fueron exitosas y luego de realizarlas la interfaz web tanto para ONOS como para OpenDaylight mostraban los hosts como se muestra en la figura 21 y 22. Como se mencionó en la sección 5.7 para ONOS fue necesario activar antes la aplicación para el reenvío de paquetes (org.onosproject.fwd)

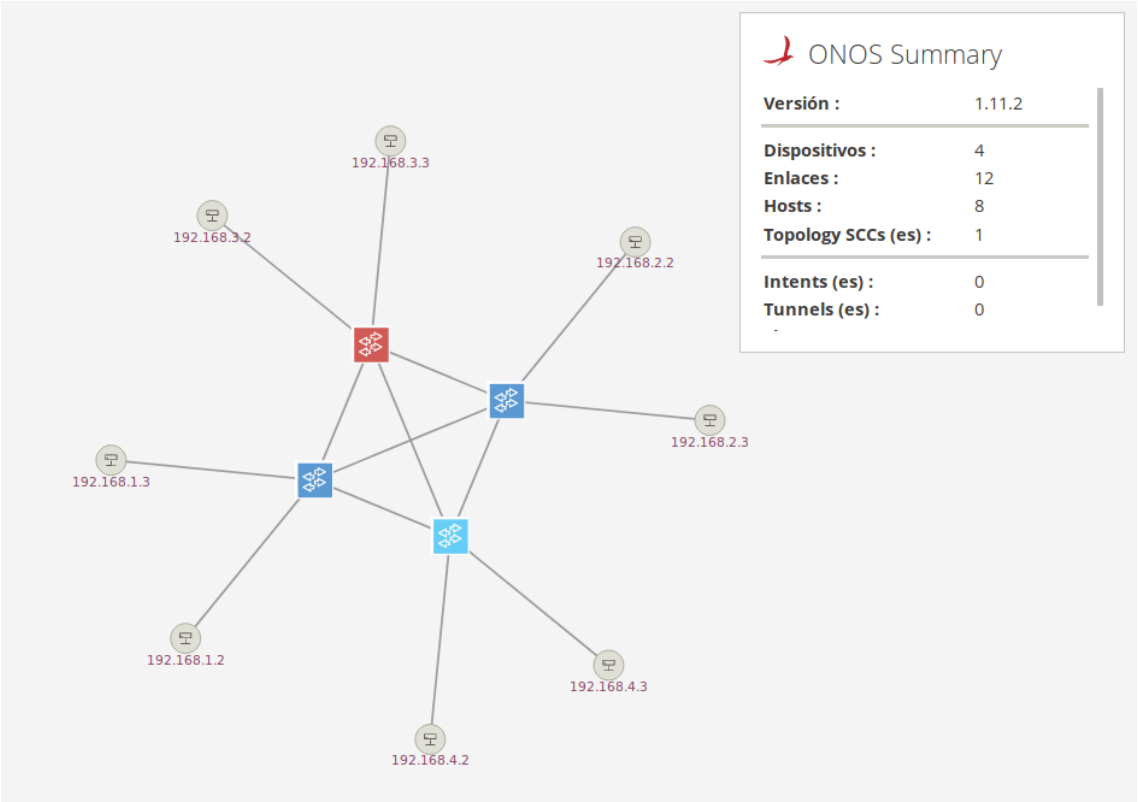

Figura 21. Topología en ONOS después de realizada la prueba de ping.

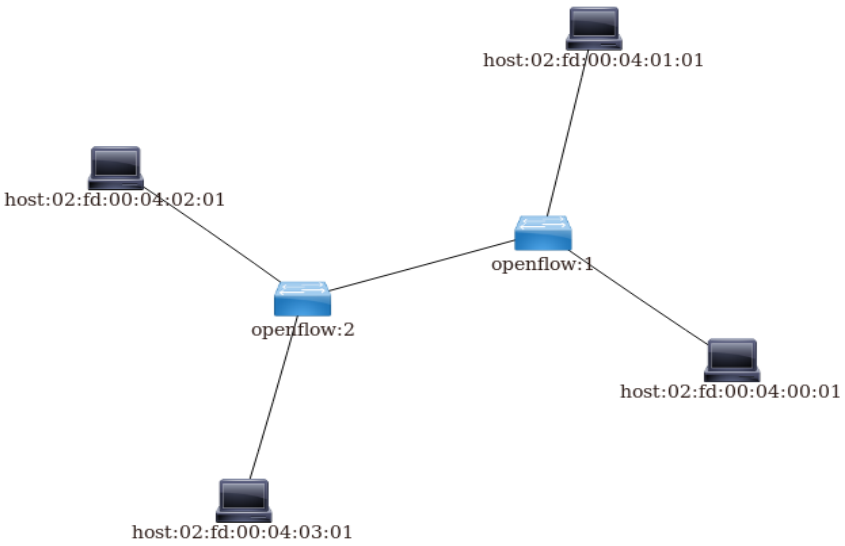

Figura 22. Topología en OpenDaylight después de realizada la prueba de ping.

8 Experimentos con Múltiples Controladores

En este capítulo se describen los experimentos de múltiples controladores realizados y sus resultados. Estas pruebas se basan principalmente en tres parámetros: Escalabilidad, disponibilidad y prestaciones.

A continuación, se define el testbed del entorno utilizado para la realización de las pruebas.

Sistema Operativo: Ubuntu 18.04 64 bits
Memoria: 16 GB
Procesadores: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
Almacenamiento: 250 GB SSD

Es importante tomar en cuenta que los resultados pueden verse afectados por tratarse de un entorno virtualizado y limitado en recursos.

Para empezar a utilizar la herramienta asegurarse de contar con los requisitos establecidos en el capítulo 6.3 y ejecutar el archivo Main.py con permiso de super usuario como se muestra a continuación:

```
sudo python3 Main.py
```

8.1 Tiempo de Descubrimiento de Topología y Máxima Escalabilidad

En esta sección se presentan los resultados del experimento de tiempo de descubrimiento de topología realizado en ONOS y OpenDaylight, se indica la cantidad máxima de conmutadores soportados por los entornos desplegados y se realiza una comparativa entre las latencias de ONOS y OpenDaylight.

8.1.1 Tiempo de Descubrimiento de Topología en ONOS

Se realizaron pruebas en los diferentes escenarios para medir la latencia al descubrir una topología en anillo, se fue incrementando la cantidad controladores y de conmutadores se obtuvieron como resultados las gráficas que se muestran en la figura 23.

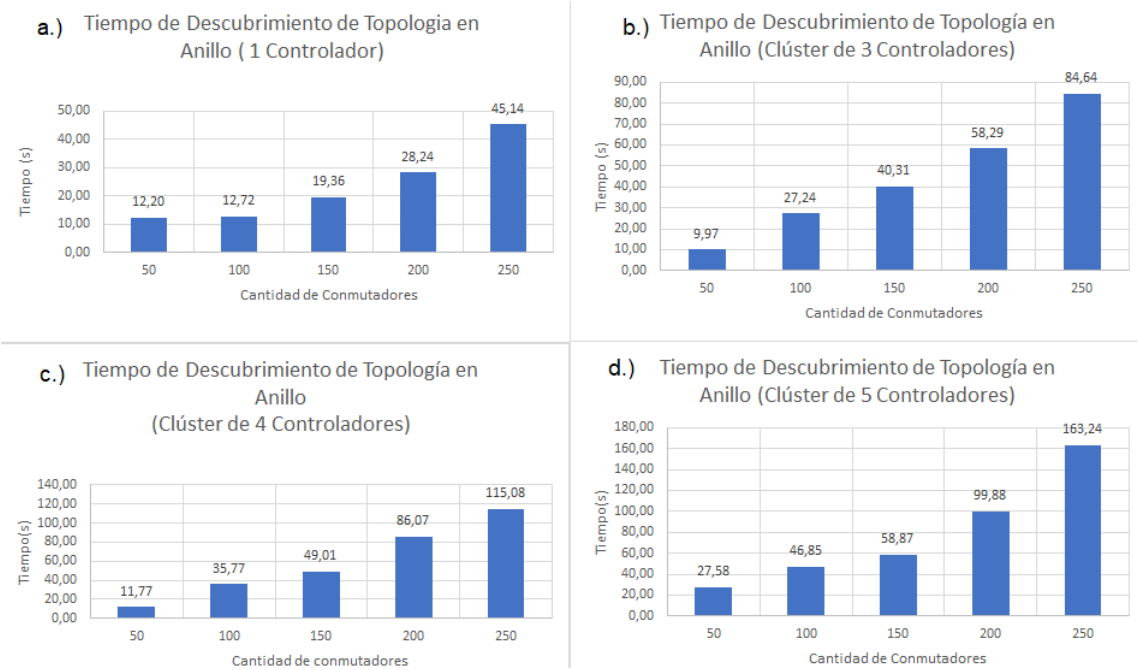

Figura 23. Gráficas tiempo de Descubrimiento en ONOS

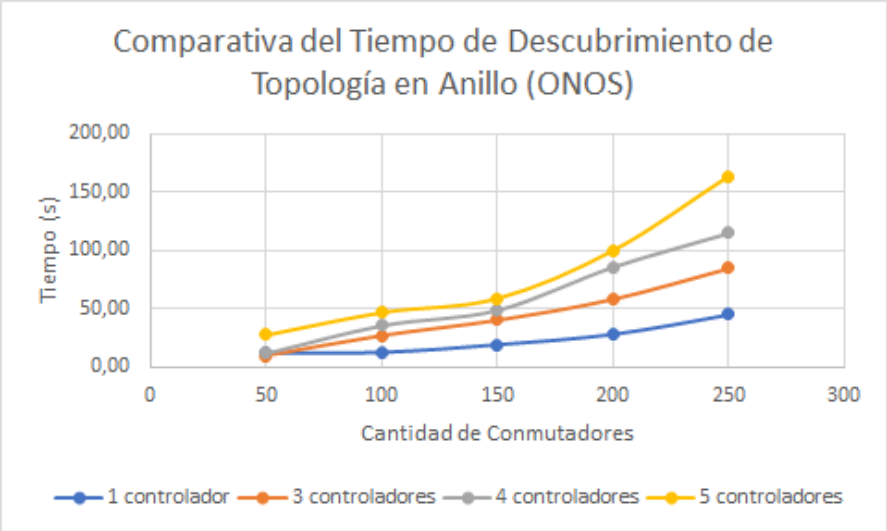

Figura 24. Comparativa del tiempo de descubrimiento en ONOS con diferentes cantidades de controladores.

8.1.2 Tiempo de Descubrimiento de Topología en OpenDaylight

A continuación, se presentan los resultados de las pruebas de descubrimiento de topología para los escenarios utilizando el controlador OpenDaylight. Se realizaron pruebas con 1 controlador y con un clúster de 3 controladores para medir la latencia al descubrir una topología en anillo, se fue incrementando la cantidad controladores y se obtuvieron como resultados las gráficas que se muestran en la figura 25.

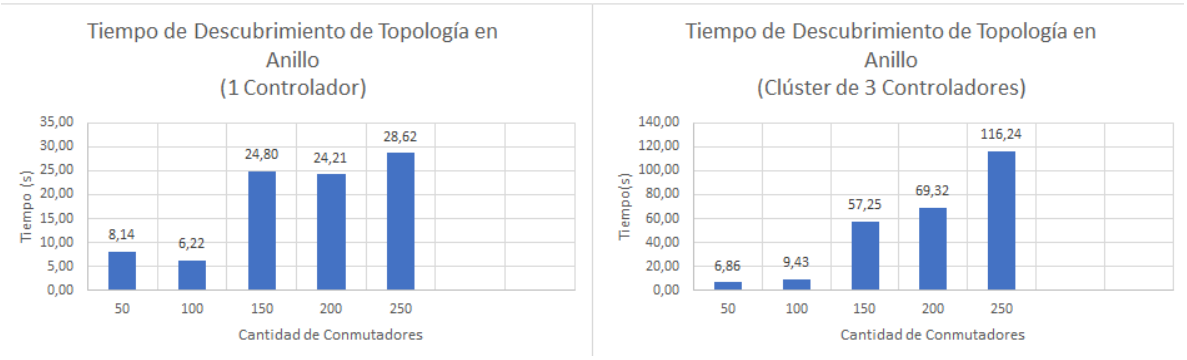

Figura 25. Gráfica de Tiempo de descubrimiento en ODL.

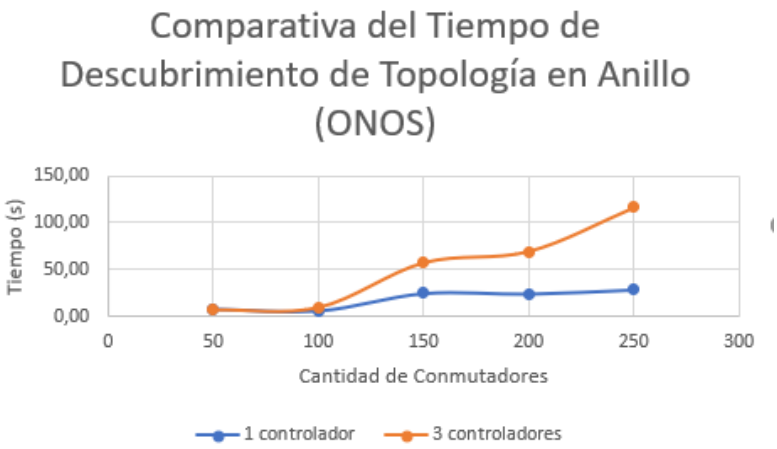

Figura 26. Comparativa del tiempo de descubrimiento en ODL con diferentes cantidades de controladores.

8.1.2 Comparativa de Tiempo de Descubrimiento de Topología en ONOS vs OpenDaylight

En el gráfico de la figura 27 se muestra una comparativa del tiempo de latencia en el descubrimiento de una topología sin clúster. Hay que tomar en cuenta que en el caso de ONOS se realiza un balance de carga que agrega tiempo de latencia.

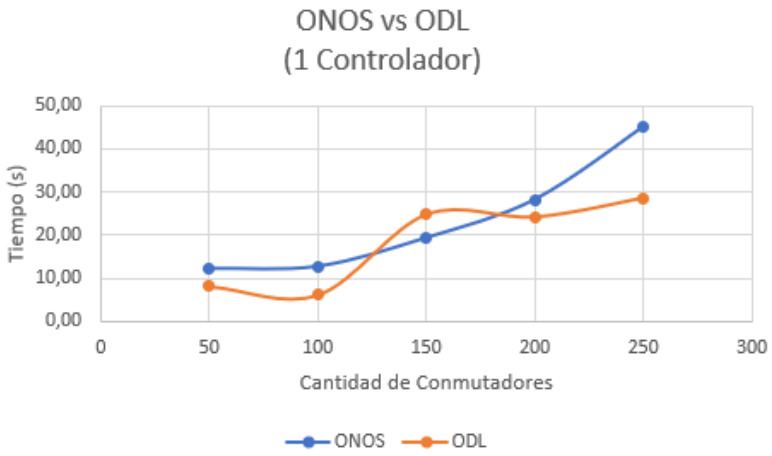

Figura 27. Comparativa del tiempo de descubrimiento de topología en ONOS y ODL en 1 controlador

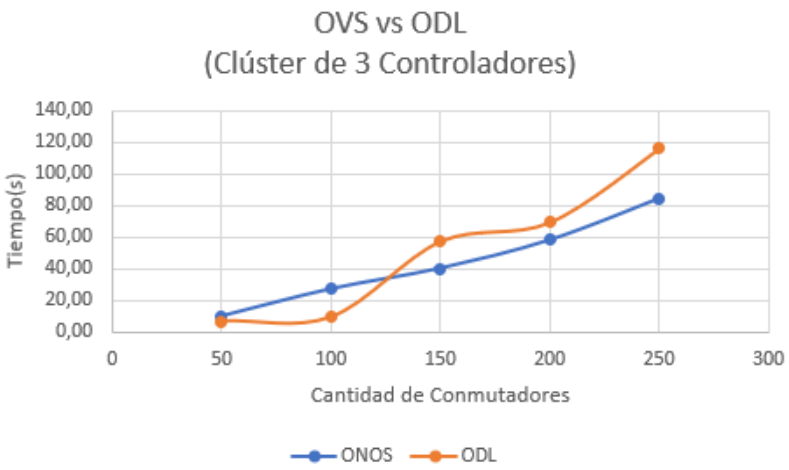

Figura 28. Comparativa del tiempo de descubrimiento de topología en ONOS y ODL en clúster.

8.2 Latencia en Eventos de Conmutadores

En esta sección se presentan los resultados para el experimento de latencia en eventos de conmutadores y puertos. Esta prueba se realizó en clústeres de 3, 4 , 5 y 6 controladores y también en un sólo controlador. Se muestran los resultados para ONOS y OpenDaylight y se realiza una comparativa de ambos.

8.2.1 Latencia en Eventos de Conmutadores NOS

En la figura 29 se observa gráficamente los tiempos de latencia para los eventos de agregar y eliminar un conmutador en diferentes entornos, utilizando el controlador ONOS. Los resultados mostrados en esta gráfica son el promedio de 10 iteraciones. Se puede constatar que los tiempos de eliminar un conmutador son menores que los de agregar.

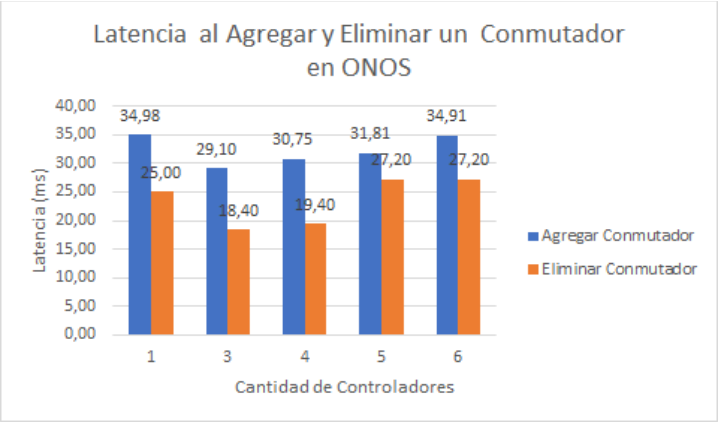

Figura 29. Latencia al agregar y eliminar un conmutador en ONOS

8.2.1 Latencia en Eventos de Conmutadores en OpenDaylight

En la figura 30 podemos observar que en OpenDaylight al igual que en ONOS el tiempo de latencia en eventos de dar de baja un conmutador es mucho menor que el de agregar uno.

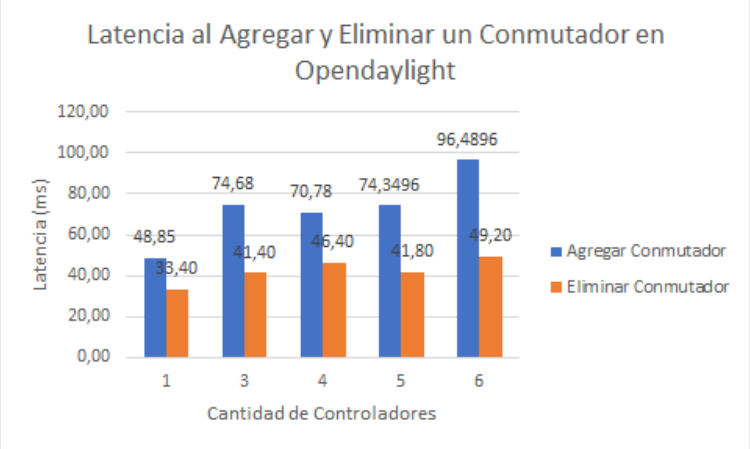

Figura 30. Latencia al agregar y eliminar un conmutador en OpenDaylight

8.2.3 Comparativa en Latencia de Eventos en ONOS vs OpenDaylight

En la figura 31 se muestra una comparación de la latencia en ONOS y OpenDaylight al agregar un conmutador y en la figura 32 al eliminar un conmutador. Como se puede observar los tiempos de latencia para ambos casos son menores ONOS. En los entornos con clúster OpenDaylight presenta el doble de la latencia en ONOS.

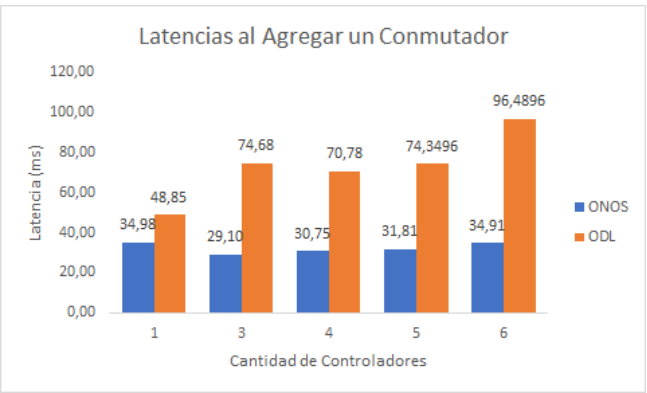

Figura 31. Latencias al agregar un conmutador

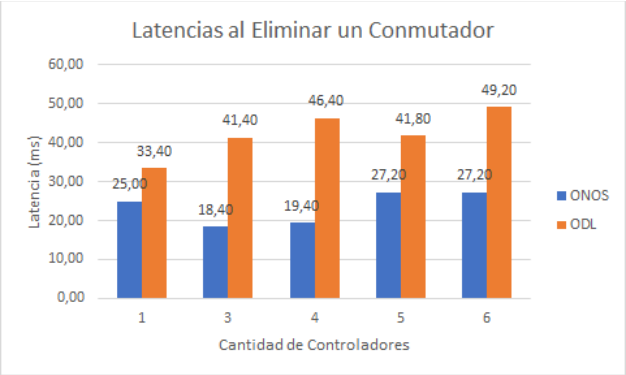

Figura 32. Latencias al eliminar un conmutador

8.3 Tolerancia a fallas

En las pruebas realizadas para verificar la tolerancia a fallas se constató que tanto los clústeres de ONOS como de OpenDaylight los sistemas se mantienen estables siempre y cuando la cantidad de controladores fallidos no sea mayoritaria, luego de dar de baja los controladores se verificó la consistencia de los datos.

8.4 Instalación de Flujos en los Conmutadores

En esta subsección se presentan los resultados de las pruebas de agregar flujos en los conmutadores desde la interfaz norte en las que se busca evaluar el impacto de contar con múltiples controladores al momento de crear flujos en la red. Las pruebas se realizaron con clúster tanto en ONOS como en OpenDaylight de 3, 4 y 5 controladores y con una topología de 9 conmutadores en anillo. Los flujos con los que se realizan pruebas son de 500, 1000, 1500 Y 2000. Se envían 10 flujos en cada solicitud realizada.

8.2.1 Instalación de Flujos en ONOS

A continuación, se muestran los resultados de las pruebas realizadas de instalación de flujos en ONOS, el procedimiento se repite enviando las solicitudes a diferentes nodos del clúster y como se observa en las figuras en ambos casos se tienen resultados muy parecidos dado que ONOS realiza un balanceo de carga y cada controlador tiene aproximadamente asignados la misma cantidad de dispositivos.

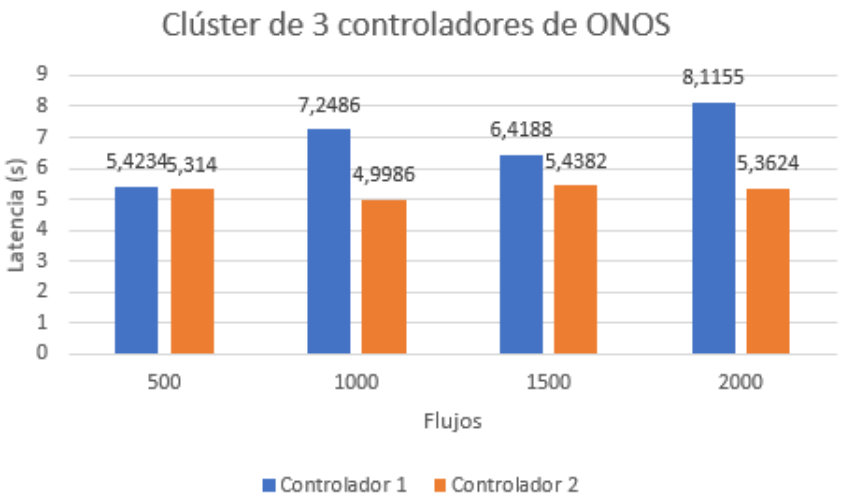

Figura 33. Instalación de Flujos en un clúster de ONOS de 3 controladores

8.3.2 Instalación de Flujos en OpenDaylight

Las pruebas realizadas en OpenDaylight de instalación de flujos no arrojan resultados certeros. En algunas ocasiones instala los flujos muy rápido en comparación con los resultados obtenidos con ONOS (500 flujos por segundo), y en otras ocasiones no completa la instalación de los flujos. Cabe resaltar que la aplicación utilizada fue desarrollada por el equipo de OpenDaylight y que con las pruebas realizadas en ONOS no se presentaron inconvenientes con la instalación de flujos.

8.5 Máxima Escalabilidad

En las pruebas realizadas en ONOS con un sólo controlador la escalabilidad máxima que se observa es de aproximadamente 1200 conmutadores, esto en una red en anillo, es decir que además soporta 1200 enlaces. Se verificó que, con un clúster de 3, 4 y 5 controladores también se soportaba los 1200 conmutadores. Según el sitio web de ONOS un clúster de 3 controladores puede soportar una topología de hasta 16000 conmutadores, pero hay que tomar en cuenta los recursos limitados y escenario virtualizado.

Para el caso OpenDaylight se fue aumentando la cantidad de conmutadores y se pudo notar que el caso de 1 solo controlador sin clúster se puede soportar hasta 600 conmutadores en anillo, al intentar escalar la cantidad de conmutadores se producen desconexiones, quedando la red inestable. Cuando se tiene un clúster de 3 nodos se soporta hasta aproximadamente 350 conmutadores, si aumentamos a 400 conmutadores podemos observar en los logs eventos de conexión y desconexión de los dispositivos, es decir, que la red no llega a mantenerse estable. Este comportamiento puede estar relacionado con los pocos recursos utilizados en las pruebas, ya que en los momentos de desconexiones se verificó con el comando "top -i" que el CPU estaba siendo usado al máximo .

9 Conclusiones y Trabajos Futuros

En los capítulos anteriores fue descrito el diseño de la herramienta utilizada, cada una de las pruebas realizadas y se mostraron los resultados obtenidos. En este apartado se realizan unas conclusiones en base a los resultados obtenidos y se plantean una serie de trabajos futuros con los que se pueda aprovechar las utilidades que hasta el momento brinda la herramienta.

9.1 Conclusiones

En este trabajo se ha presentado el desarrollo e implementación de una herramienta que permite desplegar entornos de múltiples controladores haciendo uso de herramientas de código abierto. Se utilizaron los controladores ONOS y OpenDaylight que son los más comúnmente utilizados y se realizaron experimentos buscando analizar principalmente tres parámetros: escalabilidad, tolerancia a falla y prestaciones. Es importante considerar las limitaciones de recursos para los escenarios manejados.

Con el desarrollo de esta herramienta se pudo utilizar el módulo de python TkInter y se comprobó las facilidades que ofrece para crear una interfaz gráfica, como lo es la posibilidad de organizar la interfaz por medio de cuadrículas, permitiendo así ubicar fácilmente botones, etiquetas, selectores, etc.

En lo referente al uso de ONOS y OpenDaylight se observó una mejor documentación de Opendaylight en cuanto al modo de funcionamiento del clúster de controladores. ONOS por su parte presenta una mejor documentación en lo referente a prueba de los controladores y muestra en su web los resultados de distintas pruebas en cada versión.

Por otro lado, la interfaz por la línea de comandos en ONOS brinda mayores facilidades comparada a Opendaylight. Fácilmente se puede observar la cantidad de flujos con los que cuenta cada conmutador sólo escribiendo “flows”; permite también verificar cuál es el máster de cada controlador; la cantidad de dispositivos que se encuentran conectados; los nodos que forman un clúster y el tiempo que tienen operativo; entre otras funcionalidades.

Las pruebas realizadas en cuanto a escalabilidad no arrojaron los resultados esperados; no se registró un incremento en la cantidad de conmutadores soportados a medida que se aumentaba el tamaño del clúster. Este comportamiento está relacionado con la pocos recursos del entorno utilizado para las pruebas, ya que hay una gran diferencia con los resultados presentados por ONOS en su página web. De acuerdo con la documentación en el sitio web de ONOS las pruebas realizadas por ellos se hicieron sobre un clúster de 7 servidores físicos, con 10 cores reales y 64GB de memoria.

En las pruebas realizadas en cuanto a escalabilidad ONOS presenta mejores resultados que OpenDaylight. Para un entorno sin clúster ONOS llega a soportar hasta 1200 conmutadores conectados, mientras que OpenDaylight soporta como máximo 800. En el caso de ONOS este resultado puede estar relacionado con la falta de recursos para la realización de las pruebas.

En las pruebas de descubrimiento de topología se pudo observar que a medida que se aumenta la cantidad de conmutadores que conforman un clúster, aumenta también el tiempo en el que se descubre la topología. Este hecho se debe a que cada conmutador se conectaba con todos los controladores por lo que el tiempo de conexión hacía que aumentara el tiempo total de descubrimiento de la topología.

En las pruebas de medición de latencia ante eventos como agregar un nuevo conmutador a la red o eliminarlo, se pudo notar tanto en ONOS como el OpenDaylight que eliminar un conmutador tienen una menor latencia que agregar un nuevo conmutador, esto es de gran importancia en una red para que se puede redireccionar el tráfico rápidamente en caso de falla. En esta prueba se observó también mejores prestaciones de ONOS, llegando OpenDaylight en la mayoría de los casos a tener el doble de latencia que ONOS.

ONOS y OpenDaylight tienen algunas características similares en su funcionamiento como el uso de algoritmo RAFT para el consenso dentro de un clúster, el uso de la consola Karaf, entre otros. Estos controladores pueden coexistir en el mercado, siendo ambos auspiciadas por Linux Foundation, ya que están diseñadas con diferentes enfoques y funcionalidades. ONOS se enfoca en brindar un buen rendimiento y la escalabilidad que las grandes operadoras requieren, mientras que OpenDaylight está orientado a brindar las muchas funcionalidades necesarias a nivel de un centro de datos.

9.2 Trabajos Futuros

Se ha desarrollado una herramienta funcional, aunque al realizar las pruebas se han encontrado limitaciones que llevan a sugerir los siguientes trabajos futuros:

- Es recomendable realizar las pruebas haciendo uso de mayores recursos en cuanto a CPU y memoria, para esto se propone la creación de un clúster de servidores con al menos los recursos indicados por ONOS y OpenDaylight en el testbed de sus pruebas.
- Otra línea de investigación sería llevar la aplicación a que opere en un entorno distribuido, pudiendo obtener así mayores recursos para la ejecución de pruebas más fiables. Una de las opciones es configurar EDIV (Distributed VNX (EDIV) [29]

- La herramienta está diseñada con programación modular lo que permitirá la agregar un nuevo controlador sin suponer cambios en el resto de las funcionalidades de la herramienta. Sería interesante que los controladores a agregar tengan un diseño diferente, por ejemplo, un controlador que utilice una estrategia de conexión al conmutador de IP alias. Agregar un conmutador que funcione con IP alias permitiría estudiar conceptos teóricos como que los cambios en los controladores son más rápidos porque no requiere consenso; que la recuperación ante fallas también es más rápida porque no se tiene que elegir un líder; o si realmente es más propenso a inconsistencias.
- Como se explicó en el capítulo 5 a partir de la versión 1.14 de ONOS el funcionamiento del clúster cambia y es necesario crear un clúster de Atomix que luego se conecta a ONOS. Una de las investigaciones futuras puede estar orientada a realizar pruebas con estas versiones de ONOS para determinar las ventajas agrega este nuevo modo de funcionamiento.
- La aplicación actualmente al momento de lanzar una red agrega una cantidad de conmutadores conectados entre sí ya sea por medio de una topología en malla o en anillo, por lo que otro trabajo futuro sería agregar a la herramienta la funcionalidad de elegir entre diferentes otros tipos de topologías, o bien agregar una topología personalizada como lo hace mininet.

Bibliografía

[1] Nick Feamster, Jennifer Rexford, Ellen Zegura."The road to SDN: an intellectual history of programmable networks" ACM SIGCOMM Computer Communication Review. v.44 n.2. New York, NY, USA. Abril 2014

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks". New York, USA. Abril 2008

[3] Open Networking Foundation. Disponible en: https://www.opennetworking.org

[4] Open Network Foundation. "SDN Architecture Overview". Diciembre, 2013. Disponible en: https://www.opennetworking.org/images/stories /downloads/sdn-resources/technicalreports/SDN-architecture-overview-1.0.pdf

[5] List of SDN controller software. Disponible en: https://en.wikipedia.org/wiki/List_of_SDN_controller_software

[6] Lauren Horwitz. "The seven benefits of softwaredefined networking". Cisco. Disponible en: https://www.cisco.com/c/en/us/solutions/softwa re-defined-networking/benefits.html

[7] Open Network Foundation. "Software-Defined Networking: The New Norm for Networks". Abril, 2012. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[8] Amy Larsen DeCarlo. Disponible en: Disponible en: https://searchdatacenter.techtarget.com/es/consejo/Pensando-implementar-SDN-en-su-empresa-No-tan-rapido

[9]Yustus Eko Oktian, SangGon Lee, HoonJae Lee, JunHuy Lam. Distributed SDN controller system: A survey on design choice. 2017. Disponible en: https://www.sciencedirect.com/science/article/pii/S1389128617301706

[10] Open Network Foundation. "OpenFlow Switch Specification versión 1.0". Diciembre, 2011. Disponible en: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf

[11] Open Network Foundation. “OpenFlow Switch Specification versión 1.0”. Diciembre, 2011. Disponible en: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pd

[12] V. Yazici, M.O. Sunay, and A.O. Ercan, "Controlling a software-defined network via distributed controllers. Enero, 2014. Disponible en: https://faculty.ozyegin.edu.tr/aliercan/files/2012/10/YaziciNEM12.pdf

[13] J. Hu, C. Lin, X. Li and J. Huang. "Scalability of control planes for Software defined networks: Modeling and evaluation," 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS).Hong Kong. 2014. Disponible en: https://ieeexplore.ieee.org/document/6914314?arnumber=6914314&tag=1

[14] A. Panda, C. Scott, A. Ghodsi, T. Koponen, S. Shenker. Cap for networks. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. Hong Kong. 2013. Disponible en: https://dl.acm.org/citation.cfm?id=2491186

[15] OpenDaylight Project. Disponible en: https://www.OpenDaylight.org/

[16] D. Suh, S. Jang, S. Han, S. Pack, T. Kim and J. Kwak, "On performance of OpenDaylight clustering," 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, 2016, pp. 407-410. doi: 10.1109/NETSOFT.2016.7502476. Disponible en: https://ieeexplore.ieee.org/abstract/document/7502476

[17] OpenDaylight Project. “OpenDaylight Documentation - Security Considerations”. Disponible en: https://docs.OpenDaylight.org/en/stable-fluorine/getting-started-guide/security_considerations.html#

[18] Ahmad Hemid. “Facilitation of The OpenDaylight Architecture”. Mayo 2017. Alemania. Disponible en: https://www.researchgate.net/publication/317057083_Facilitation_of_The_OpenDaylight_Architecture

[19] OpenDaylight Project. “ODL Plataform Overview”. Disponible en: https://www.OpenDaylight.org/what-we-do/odl-platform-overview

[20] ONOS Project. Disponible en: https://wiki.onosproject.org/

[21] ONOS Project. “ONOS Platform Architecture”. Disponible en:https://www.slideshare.net/OpenDaylight/onos-platform-architecture

[22] S. Secci, S. Scott-Hayward, Q. Pham Van, D. Verchere, A. Sow, C. Basquin, D. Smyth, K. Attou, K. Thimmaraju, A. Campanella # , "ONOS Security & Performance Analysis (Report No. 2)", Informational Report, Open Networking Foundation, November 20180

[23] Python. Disponible en: https://www.python.org

[24] Virtual Networks over linuX (VNX). Disponible en: http://web.dit.upm.es/vnxwiki/index.php/Main_Pag

[25] Whireshark. "Tshark". Disponible en: https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html

[26] OpenDaylight Project. "Running and testing an OpenDaylight Cluster". Disponible en: https://wiki.OpenDaylight.org/view/Running_and_testing_an_OpenDaylight_Cluster

[27] Python. "TkInter". Disponible en: https://wiki.python.org/moin/TkInter

[28] ONOS Project. "Test Plans". Disponible en: https://wiki.onosproject.org/display/ONOS/Test+Plans

[29] VNX. "Ediv-install". Disponible en: http://web.dit.upm.es/vnxwiki/index.php/Ediv-install