PLATAFORMA DE APLICACIÓN DE POLÍTICAS DE SEGURIDAD PARA EL CONTROL DE TRÁFICO EN REDES DEFINIDAS POR SOFTWARE

LUCAS VÁSQUEZ AGUDELO

Ingeniería de Telecomunicaciones

MSc. JUAN CAMILO CORREA CHICA

INSTITUTO TECNOLÓGICO METROPOLITANO

MAYO 03 DEL 2018

RESUMEN

Las redes definidas por software (SDN) es quizás el cambio de paradigma más importante en los últimos tiempos en el campo de las redes de datos. *SDN* separa el plano de control (*Encargado de decidir cómo se maneja el tráfico de red)* del plano de datos *(Encargado de conmutar el tráfico de red de acuerdo a las políticas establecidas en el plano de control)*, esta separación se lleva a cabo colocando el control en un servidor centralizado (Controlador SDN). Uno de los desafíos principales que se exponen con la inclusión de esta nueva arquitectura *SDN* es la seguridad, donde los firewalls son los dispositivos encargados de asegurar el tráfico que circula en la red de acuerdo a las políticas de seguridad creadas por los ingenieros de red.

En el presente documento se presenta una plataforma de firewall que se aplica a un controlador *OpenDaylight* SDN, capaz de traducir las políticas de redes creadas por el administrador de la red en acciones que llevan a cabo los dispositivos del plano de datos, mediante el protocolo *OpenFlow.*

La plataforma es implementada en un escenario de simulación virtual controlado, usando las herramientas de software *Oracle VM VirtualBox, Mininet y* un controlador *ODL.*

Los resultados obtenidos evidencian que la plataforma para aplicar políticas de seguridad para el control de tráfico es una buena herramienta que se presenta de forma intuitiva para los ingenieros y operarios de la red, ya que permite ingresar dichas políticas de forma rápida y sencilla y ser aplicadas por el controlador *SDN OpenDaylight* de forma rápida y eficaz, a partir de los resultados obtenidos se desdobla que la implementación puede ser aplicado en otros escenarios no virtuales y utilizando equipos físicos reales que soporten el protocolo *OpenFlow.*

Palabras claves*: *Software Defined Network, OpenFlow, OpenDaylight, OpenvSwitch, Middlebox, Mininet, Firewall, Políticas de red.

RECONOCIMIENTOS

Quiero destacar y reconocer a varios actores que influyeron en mi proceso de formación académico e integral durante estos años de carrera profesional. Principalmente a mi familia que han sido mi motivación, especialmente a mi tía que siempre está conmigo sin importar las circunstancias a ella le dedico mis logros, a mi padre agradecerle su esfuerzo constante por darme lo mejor y a mis hermanos que comparten todo conmigo.

A mis amigos y futuros colegas profesionales del ITM que siempre hacen las clases más divertidas, a mis docentes a lo largo de mi carrera, especialmente a Juan Camilo Correa Chica que es un excelente director de grado y maestro ejemplar.

ACRÓNIMOS

En esta sección se presentan los acrónimos empleados a lo largo de este documento, con el objetivo de facilitar la comprensión del contenido por parte del lector.

SDN: Software Defined Network.

OP: OpenFlow Protocol.

PD: Plano de Datos.

PC: Plano de Control.

ODL: OpenDayLight.

API: Application Programming Interface.

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

1. INTRODUCCIÓN

1.1 GENERALIDADES

El continuo crecimiento de internet ha influido sobre casi todos los ámbitos de nuestra sociedad, cada día surgen nuevos dispositivos conectados a la red y con ellos se pone a nuestra disposición nuevos servicios y aplicaciones, esto ha obligado a expandir y reforzar la infraestructura de las redes para poder suplir los requisitos que impone las demandas.

Dicho esto, la infraestructura de redes se ha vuelto bastante compleja, difícil de gestionar y administrar ya que se incluyen múltiples dispositivos de redes *(Middlebox)*, cada uno maneja su propio sistema operativo y este por lo general es propietario y de código cerrado por cada fabricante. En la arquitectura de red actual, los planos de control y de datos están ligados y distribuidos en cada equipo de la red, cuando un paquete entra a un *Middlebox* este consultará su propio plano de control para decidir qué hacer con el paquete, este proceso se repite en cada dispositivo de red intermedio entre el origen y el destino, debido a esto los ingenieros de redes deben configurar generalmente de forma individual cada *Middlebox* utilizando diferentes interfaces de gestión y administración que varía dependiendo del fabricante, este modo de operación ha aumentado la complejidad de gestión, administración y los costos operacionales de la red.

Por otro lado, la seguridad de la red es de vital importancia para cualquier organización ya que garantiza la confiabilidad, integridad y disponibilidad de los datos y servicios dentro de la red. Los *firewalls* son los dispositivos encargados de analizar los flujos de datos y aplicar políticas de seguridad para prevenir y mitigar las amenazas dentro de la red, por esta razón es que estos dispositivos desarrollan un papel importante en toda la infraestructura. La gran limitación que tienen es que así mismo como los enrutadores y conmutadores, los *firewalls* están distribuidos a lo largo de la red y también los planos de datos y control están ligados, lo que hace que su administración y modo de operación sea poco escalable y eficiente al momento de configurar o aplicar ciertos cambios.

Así mismo en las redes tradicionales los *firewall* se ubican en locaciones estratégicas de la infraestructura de la red, generalmente en el perímetro, donde termina la *intra-net* y donde empieza la *extra-net* con el fin de analizar el flujo de datos y comparar la información de los encabezados de los paquetes con unas reglas de flujo previamente establecidas con el fin de denegar o permitir el tráfico según sea el caso, este modelo es poco escalable y rígido, debido a que si la red se expande los firewalls deberán ser reubicados físicamente y nuevas políticas de red tendrán que ser creadas en cada uno de ellos, es decir su dificultad para adaptarse a entornos cambiantes es su principal falencia.

Como se exponen en (Diego Kreutz, 2014) *y* (Nick Feamster, 2014) las redes definidas por software (*SDN, Software Defined Network*) se presentan como una solución a esta problemática, cambiando la forma en que se diseñan y administran las redes tradicionales, *SDN* separa el plano de control (Decide cómo manejar el tráfico) del plano de datos (Reenvía el tráfico de acuerdo a las decisiones del plano de control) de modo que con la separación de estos planos los conmutadores y enrutadores se convierten en equipos de reenvío simples y la lógica de control se implementa en un servidor centralizado denominado Controlador *SDN* con el fin de gestionar todos los dispositivos de la red y hacer las redes más programables y con una administración centralizada.

Con la introducción del concepto *SDN* las redes se vuelven programables a través de aplicaciones que se ejecutan en la interface de la parte superior del controlador *(Northbound Interface)*, las cuales permiten definir políticas de redes por medio de lenguajes de alto nivel en lugar de configuraciones especificas en los *Middleboxes*, con la ayuda de la interface inferior del controlador *(Southbound Interface),* es posible la comunicación entre los dispositivos del plano de datos con el controlador *SDN*, permitiendo ejecutar las políticas de red definidas en la aplicación.

Expuestos los puntos anteriores se hace evidente que el controlador *SDN* es el ente informático que realiza todas las funciones de control, y contar con un soporte de seguridad hará posible garantizar el funcionamiento de la red, en este documento se propone un prototipo de una *API (Application Programming Interface)* de *firewall* capaz de traducir políticas de seguridad de red en acciones puntuales para el control del tráfico en *switches SDN*, con el fin de tener un sistema de

firewall centralizado y programable que se adapte ágilmente a los cambios que puedan presentarse en la red, con una interfaz intuitiva para el usuario. El funcionamiento será evaluado dentro de un entorno de simulación virtual controlado.

1.2 OBJETIVOS

1.2.1 General

Obtener un prototipo de *firewall* para redes definidas por *software* que permite traducir políticas de seguridad, diseñadas en lenguaje de usuarios y operadores de red, en acciones de detección, mitigación y control de tráfico en dispositivos de red programables.

1.2.2 Específicos

- Diseñar una aplicación de red que permita instruir políticas de seguridad a un controlador *SDN* usando un lenguaje e interfaz intuitivos para usuarios y operadores de red.

- Definir una interfaz que permita establecer un protocolo de comunicación entre la capa de aplicación y la capa de control *(Northbound API).*

- Desarrollar un mecanismo de capa de control que permita traducir políticas de seguridad en reglas de flujo para los dispositivos en la capa de datos de la red *SDN*.

- Evaluar el funcionamiento de las políticas de seguridad en un entorno de simulación controlado.

1.3 ESQUEMA DEL TRABAJO DE GRADO

En este apartado se muestra la distribución del presente informe con el objetivo de dar un orden lógico al contenido del trabajo, éste está comprendido por 5 capítulos que se describirán a continuación:

1.3.1 Capítulo 1. Introducción

El capítulo uno consiste en presentar de manera resumida el problema abordado, su justificación y objetivos planteados para el desarrollo del proyecto.

1.3.2 Capítulo 2. Marco Teórico

En este capítulo se exponen los conceptos, teorías y tecnologías relevantes involucradas en este proyecto, además se resaltan los documentos, publicaciones y revistas científicas en el apartado del estado del arte relacionado con el foco principal del proyecto.

1.3.3 Capítulo 3. Metodología

En esta sección se condensa toda la información de forma detallada relacionada con la metodología empleada para el desarrollo de cada uno de los objetivos específicos para alcanzar el objetivo principal del proyecto.

1.3.4 Capítulo 4. Resultados y Discusión

En el capítulo 4 se presentan los resultados de forma concisa y detallada obtenidos de cada uno de los objetivos trazados durante la ejecución del proyecto, posteriormente expondrá sus fortalezas y limitación de la metodología y los resultados obtenidos.

1.3.5 Capítulo 5. Conclusiones y trabajo a futuro

Con este capítulo se finaliza el trabajo donde se ilustran las conclusiones de trabajo realizado, las recomendaciones a tener en cuenta y se propone algunos posibles escenarios para continuar con un plausible trabajo a futuro.

2. MARCO TEÓRICO

En el presente capítulo se ilustra el grupo central de conceptos necesarios para poder comprender el tema principal y posteriormente poder desarrollar el proyecto final, además se presentan un apartado de estado del arte donde se exponen algunos documentos encontrados a lo largo de la investigación relacionados con el tema central del trabajo.

2.1 Redes Definidas por Software

La *ONF (Open Networking Foundation)* es una comunidad compuesta por varias comunidades, las cuales trabajan en conjunto para llevar a cabo diferentes proyectos. La *ONF* fue la organización que comenzó el movimiento de las redes definidas por software (*SDN,* por sus siglas en inglés) en el 2011, las cuales se presentan como una tecnología emergente que permite tener arquitecturas de red con un marco programable, e igualmente, constituyen un nuevo esquema que propone el desacople del plano de control *(PCL)* respecto al plano datos *(PDS),* permitiendo que el control se implemente de forma centralizada en la red mediante un controlador instalado en un servidor el *PCl,* con el objetivo de mantener una vista global de la red con la capacidad de controlar varios dispositivos de reenvío como enrutadores y conmutadores lo cual corresponde al plano de datos (OpenNetworking, 2018). En esa medida, el desligamiento de estos planos hizo necesaria la introducción de un nuevo protocolo de comunicaciones llamado *OpenFlow Protocol (OP)* el cual abordaremos más adelante en este mismo capítulo; Este nuevo ambiente promete a los ingenieros de red administrar el flujo de datos de una manera dinámica para suplir las demandas de las redes y servicios actuales, *SDN* está fundamentado en software *opensource* lo cual facilita la configuración, gestión, aseguramiento y optimización de la red mediante la creación de programas y aplicaciones *SDN* que podrán ser desarrollados por los administradores e ingenieros de redes y ser aplicado sin ningún inconveniente.

Estas aplicaciones y programas *SDN* son creadas de acuerdo con las necesidades propias de la red y el funcionamiento deseado, aplicaciones pueden ser ingresadas directamente a un controlador a través de las Northbound *API (Application Programming Interface),* a su vez el controlador se

comunicará con el plano de datos a través de las *Southbound API,* un claro ejemplo de una *Southbound API,* es el protocolo *OpenFlow* (Lara, Network Innovation using OpenFlow: A Survey, 2014).

La aceptación y recepción de este nuevo paradigma por parte de la comunidad de redes ha aumentado de forma gradual, organizaciones importantes como Google, Amazon han desplegado parte de su infraestructura basándose en el esquema SDN, El impulso de SDN fue lo suficientemente fuerte como para hacer que *Google, Facebook, Yahoo, Microsoft, Verizon y Deutsche Telekom* financiaran *Open Networking Foundation* (Sanchez-Velazquez, 2017, June).

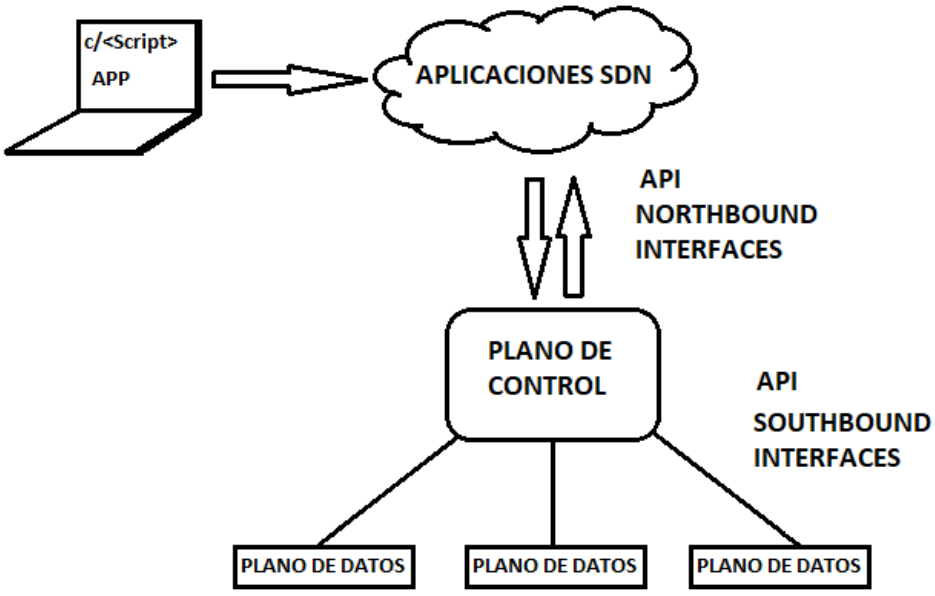

Ilustración 1. Arquitectura SDN. Fuente: Autor

2.2 Protocolo de comunicación OpenFlow.

El protocolo *OpenFlow (OP)*, cumple un papel fundamental en el esquema *SDN* ya que su principal función es proporcionar el puente de comunicación entre el controlador *SDN* y los dispositivos del plano de datos (switches, enrutadores, entre otros), sin la necesidad de que los proveedores expongan el código de sus dispositivos.

OpenFlow modifica la arquitectura de red tradicional en el sentido de que los elementos del plano de datos se convierten en dispositivos simples de reenvío de paquetes según las políticas dadas por el controlador (Lara, Network Innovation using OpenFlow: A Survey, 2014).

Desde los inicios de las redes definidas por software, *OpenFlow* ha sido la primera interfaz estandarizada entre los controladores *SDN* y los conmutadores *SDN,* ofreciendo soporte para diferentes protocolos comúnmente usados que van desde la capa 2 a la capa 4 de *OSI*. (ONF, 2016, p. 8)

OpenFlow estandarizó inicialmente un modelo del plano de datos y una *API* del plano de control basándose en que los conmutadores ya son compatibles con *OP*. Específicamente, debido a que los conmutadores de red ya admitían control de acceso y control de flujo con gran precisión, habilitar el conjunto inicial de capacidades de *OpenFlow* en un conmutador es tan fácil como realizar una actualización de *firmware*; es decir que los proveedores y fabricantes no necesitan actualizar el hardware para hacer que sus switches admitieran *OP* (Feamster, Rexford, & Ellen Zegura, 2014).

OpenFlow se describe como un protocolo abierto para permitir que las aplicaciones de software programen la tabla de flujo de diferentes conmutadores, este concepto se abordará más adelante; *OpenFlow* ha sido diseñado para apoyarse de tres componentes, un conmutador *OpenFlow* ya sea virtual o físico, un controlador *SDN* y un canal seguro para la comunicación entre el switch y el controlador generalmente *TLS (Transport Layer Security) y SSL (Secure Socket Layer*) (Lara, Network Innovation using OpenFlow: A Survey, 2014)**.**

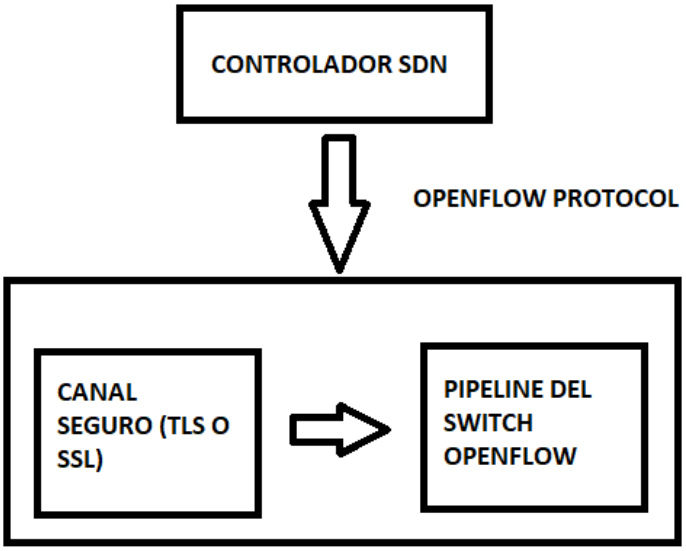

Ilustración 2. Componentes Protocolo Openflow. Fuente: Autor

Para que el controlador *SDN* pueda gestionar y configurar los dispositivos de la red por medio del protocolo *OpenFlow* se emplea tres tipos de mensajes cada uno con múltiples subtipos en la siguiente tabla se describen claramente (Open Networking Foundation, 2015, pp. 38 - 42).

- **Controlador-Conmutador:** Este tipo de mensajes son emitidos por el controlador y son usados para gestionar o indagar el estado de algún dispositivo en la red, estos mensajes pueden tener o no respuesta según sea el propósito del mensaje.
- **Asíncronos:** Son los mensajes iniciados por el dispositivo del plano de datos y van dirigidos hacia el controlador y son usados para informar sobre una eventualidad en la red o sobre el estado propio del dispositivo emisor.
- **Simétricos:** Son mensajes que inicia el controlador o el dispositivo de la red *SDN,* sin la necesidad de una petición por alguna de las partes.

Tipos de mensajes OpenFlow	Subtipos de mensajes OpenFlow
Controlador-Conmutador	**Features:** El controlador puede solicitar las capacidades básicas y la identidad de un conmutador, a lo que el conmutador debe responder con una respuesta donde especifique sus capacidades junto con su identidad. Esto se realiza comúnmente al establecer el canal *OpenFlow.* **Configuration:** Con este tipo de mensaje el controlador puede establecer y consultar parámetros de configuración en el conmutador. El conmutador solo responde cuando es una consulta del controlador. **Modify-State:** El controlador envía estos mensajes para administrar el estado de los conmutadores. Su propósito principal es agregar, eliminar y modificar las entradas de flujo en las tablas de flujo dentro del *switch OpenFlow* y establecer las propiedades del puerto del *switch*. **Read-State:** Son utilizados por el controlador para recopilar información del conmutador, como la configuración actual, las estadísticas y sus capacidades. **Packet-out:** Son usados para enviar paquetes desde un puerto específico en el *switch* y para reenviar paquetes recibidos a través de los mensajes de *Packet-In*. Los mensajes deben contener un paquete o una *ID* de *buffer* que haga referencia a un paquete almacenado en el conmutador. El mensaje también debe contener una lista de acciones que se aplicarán en el orden en que se especifican; demás de una lista de acciones para ser aplicadas al paquete. **Barrier:** el controlador utiliza este tipo de mensajes para garantizar que se cumplen las dependencias entre los mensajes o para recibir notificaciones de las operaciones completadas. **Role-Request:** Usados para establecer el rol de su canal *OpenFlow,* establecer su *ID* de Controlador o consultarlos. Es más útil cuando el *switch* se conecta a múltiples

	controladores. **Asynchronous-Configuration:** Se usan para establecer un filtro para los mensajes asíncronos que el controlador desea recibir en su canal, también es más útil cuando el conmutador se conecta a múltiples controladores.
Asíncronos	**Packet-in:** Permite transferir el control de un paquete desde el conmutador al controlador, para que decida que debe hacer con el paquete. **Flow-Removed:** Informa al controlador sobre la eliminación de una entrada de flujo perteneciente a alguna tabla de flujo. Se generan como resultado de una solicitud de eliminación de flujo del controlador o cuando se excede uno de los tiempos de permanencia de la entrada de flujo en la tabla. **Port-Status:** Usados para informar al controlador de un cambio en un puerto. Por ejemplo, cuando un enlace esta caído. **Role-Status:** Informar al controlador de un cambio en su rol. Cuando un nuevo controlador es elegido como maestro. **Controller-Status:** Informar al controlador cuando cambia el estado de un canal OpenFlow. **Flow-Monitor:** Informar al controlador de un cambio en una tabla de flujo.
	Hello: Son los mensajes que se intercambian entre el interruptor y el controlador al inicio de la conexión. **Echo:** Son enviados desde el conmutador o el controlador, y deben devolver una respuesta. Se usan principalmente para verificar la vitalidad de una conexión y también se pueden usar para medir la latencia o ancho de banda.

Simétricos	**Error:** El conmutador o el controlador utilizan mensajes de error para notificar problemas al otro lado de la conexión. En su mayoría, son utilizados por el interruptor para indicar una falla de una solicitud iniciada por el controlador. **Experimenter:** Estos mensajes proporcionan una forma estándar para que los conmutadores OpenFlow ofrezcan funcionalidad adicional es un campo de preparación para las características destinadas a futuras revisiones de OpenFlow.

***Tabla 1**. Mensajes protocolo Openflow. Fuente: Autor*

2.2.1 Tablas de flujo

Los switches contienen múltiples tablas de flujo *(Pipeline),* utilizadas para el reenvío de paquetes, cada tabla de flujo a su vez está compuesta por varias entradas de flujo, cada entrada contiene campos de coincidencia, instrucciones y contadores, dicho esto los paquetes entrantes en el dispositivo se comparan con los campos de coincidencia de cada entrada, si hay un match se ejecuta la acción contenida en esa entrada, una lista de acciones puede ser por ejemplo, descartar, inundar, reenviar hacia una interfaz en particular, modificar un campo de encabezado , o enviar el paquete al controlador, los contadores se utilizan para llevar un seguimiento estadístico de los paquetes, el paquete también podrá ser encapsulado y enviado al controlador para ser procesado por él, en la siguiente imagen se ilustra este concepto de manera más clara (Open Networking Foundation, 2015, pp. 18-37).

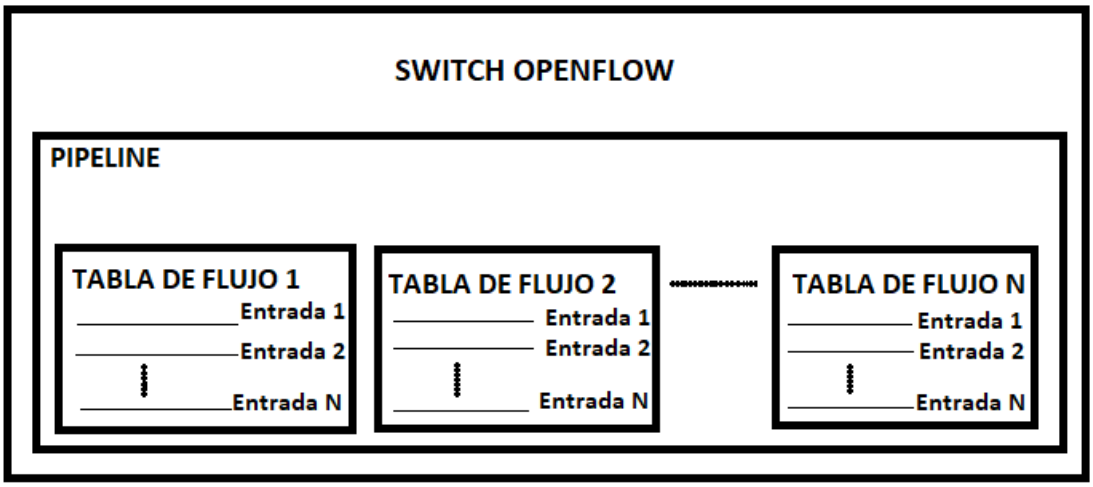

Ilustración 3. Pipeline, tablas de flujo y entradas de flujo dentro de un switche Openflow. Fuente: Autor.

2.3 Controlador SDN

El controlador es el eje central de la red *SDN* actuando como un punto de control cuya responsabilidad es manipular las tablas de flujo de los dispositivos del plano de datos, a través del protocolo *OpenFlow (API Southbound)* el controlador gestiona los dispositivos, recibe y envía información por medio de un canal seguro. Un conmutador OpenFlow deberá ser capaz de reenviar paquetes de acuerdo con las reglas definidas en la tabla de flujo, internamente el switch utiliza su *RAM (Random Access Memory) y TCAM (Ternary Content-Addressable Memory)* para tratar cada paquete (Lara, Network innovation using openflow: A survey., 2014, pp. 495 - 500)**.** Para el desarrollo de nuestra práctica nosotros usaremos *OpenDaylight* como controlador *SDN,* donde dedicaremos en una sección más adelante para explicar sus características y exponer el por qué fue el controlador elegido*.*

En la actualidad existen diferentes controladores para orquestar los dispositivos dentro de la red SDN, una especie de batalla se libra entre los fabricantes de equipos que quieren desarrollar sus controladores para orquestar sus propios equipos, y las organizaciones y fundaciones opensource que desarrollan controladores diseñados para que todos los proveedores y equipos los admitan,

(SDxCentral, 2018), a continuación, se presenta una tabla donde se muestran algunos de los controladores más destacados tanto de código abierto como propietario.

CONTROLADOR SDN	CÓDIGO FUENTE
NOX	Abierto
POX	Abierto
RYU	Abierto
FLOODLIGHT	Abierto
BEACON	Abierto
OPENDAYLIGH	Abierto
Juniper Contrail	Propietario
CISCO APPLICATION CENTRIC INFRASTRUCTURE (ACI) Y APPLICATION POLICY INFRASTRUCTURE CONTROLLER (APIC)	Propietario
VIRTUAL APLICATION NETWORKS (VAN) SDN CONTROLLER/ VIRTUAL CLOUD NETWORKING (VCN).	Propietario
PREXXI CONTROL	Propietario

Tabla 2. Tipos de controladores SDN. Fuente: Autor.

2.4 Mininet

Mininet es un emulador de red, que ofrece un entorno virtual de prueba para el desarrollo de redes *SDN*, crea una red de host, conmutadores, controladores y enlaces virtuales para interconectar los elementos mencionados; Las topologías de *Mininet* ejecutan código real bajo el *kernel* de *Linux*, lo que permite un desarrollo económico por ser software libre, puede ser ejecutado por cualquier computadora (OpenNetworkingFoundation, 2018), haciéndola una

herramienta bastante útil para nuestro interés en el desarrollo del proyecto, algunas de sus ventajas se presentan a continuación:

- Fácil instalación sobre cualquier computadora portátil o PC, en el sitio web http://mininet.org/download/#option-1-mininet-vm-installation-easy-recommended hay disponible una *VM (Virtual Machine)* para ser instalada en los *hypervisores Vmware* o *VirtualBox* tanto para los sistemas operativos *MAC, Windows o Linux*.

- Permite realizar pruebas con topologías complejas sin la necesidad de lidiar con cables para su conexión.

- Contiene una *CLI (Command Line Interface)* fácil de utilizar, pero también proporciona una *GUI (Graphical User Interface)* para la creación de topologías de red.

- Proporciona una *API (Application Programming Interface*) de Python directa y extensible para la creación y experimentación de redes.

- Los switches virtuales dentro del *Mininet* como el *switch Open vSwitch* tienen soporte para el protocolo *OpenFlow* que es el *switch* con el que vamos a trabajar dentro del proyecto.

- Otra ventaja que presenta *Mininet* mencionada anteriormente es que los hosts dentro de la topología ejecutan código real bajo el *kernel* del sistema operativo *Linux* permitiendo migrar el esquema a un entorno real sin mayores complicaciones (Mininet Team, 2018).

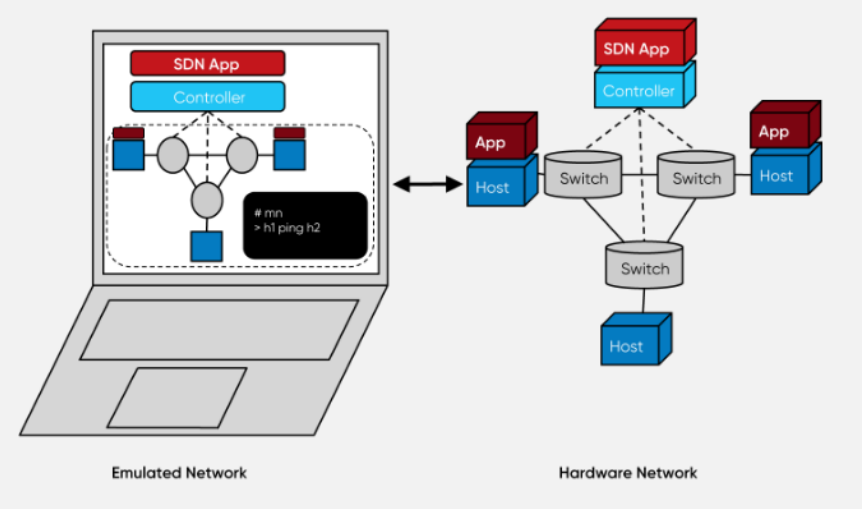

Ilustración 4. Esquema de Mininet. Fuente: https://www.opennetworking.org

2.5 Open vSwitch

Open vSwitch es un conmutador virtual multicapa con licencia de código abierto *Apache 2.0*, Este conmutador virtual es la implementación más popular de un conmutador virtual *open source* con soporte para el protocolo *OpenFlow,* y viene predefinido dentro de *Mininet,* convirtiéndose rápidamente en una parte fundamental para realizar proyectos con *SDN,* por estos motivos será el dispositivo utilizado en el presente trabajo (Čejka, 2016).

Según las especificaciones descritas en (A Linux Foundation Collaborative Project, 2016) *Open vSwitch* es un conmutador bastante completo con soporte para diferentes protocolos existentes; A continuación, se presentan algunas de sus principales características:

- Soporte para el protocolo *OpenFlow* (incluyendo algunas extensiones para virtualización).

- Admite el estándar *IEEE 802.1Q* para la encapsulación de *VLAN's (Virtual Local Area Network)* en los enlaces troncales.

- Permite el estándar *IEEE 802.1D* para la implementación de redundancia *LAN* evitando la creación de loops en la capa 2 de *OSI (STP y RSTP, Spanning-Tree Protocol* y su versión *Rapid-STP*).

- Múltiples protocolos de *tunneling* como (*GRE, VXLAN, STT* con soporte para *IPsec*).

2.6 OpenDaylight

El controlador *OpenDaylight (ODL),* es una plataforma de código abierto para redes *SDN,* la cual usa protocolos abiertos, cuyo objetivo principal es proporcionar control programático y centralizado con capacidades para monitorear dispositivos de redes físicos y virtuales dentro de la red, *ODL* al igual que muchos otros controladores, es compatible con el protocolo *OpenFlow*. Es extremadamente útil comprender que la configuración de su entorno de red con *OpenDaylight* no es una instalación de *software* única. Si bien su primer paso cronológico es instalar *OpenDaylight*, instala funciones adicionales empaquetadas como funciones de *Karaf* para satisfacer sus necesidades específicas (OpenDaylight Project, 2016-2018).

Las principales distinciones de *OpenDaylight* en comparación con otras opciones de controladores *SDN,* es la implementación de una arquitectura de micro servicios, en la cual un "micro servicio" es un protocolo, servicio o aplicación en particular que un usuario quiere habilitar dentro del controlador *ODL*, por ejemplo:

- Un complemento que proporciona conectividad a dispositivos a través de los protocolos *OpenFlow o BGP (Border Gateway Protocol).*

- Instalación de un *L2-Switch* o un servicio como *Autenticación, Autorización y Auditoria (AAA).*

- Soporte para una amplia y creciente gama de protocolos de red más allá de *OpenFlow*, incluidos *SNMP (Simple Network Management Protocol), NETCONF, OVSDB (Open V Switch Data Base)*, *BGP* entre otros.

- Soporte para desarrollar nuevas funcionalidades compuestas por protocolos y servicios de red adicionales.

Como se menciona en (OpenDaylight Project, 2016 - 2018) *Apache Karaf* proporciona un conjunto de micro servicios para ser instalados en el controlador *ODL,* inicialmente este no tiene micro servicios previamente instalados, esto con el fin de proporcionar a los ingenieros y administradores de redes un controlador personalizable, que se adapte a las necesidades de la red, expandir las capacidades del controlador y además optimizar los recursos de hardware, motivos por el cual nos interesamos en utilizar dicha herramienta en nuestro trabajo.

Según todas las indicaciones descritas (OpenDaylight Project, 2016 - 2018)**,** también es el controlador de red de código abierto más ampliamente implementado, en redes que soportan a más de 1billon usuarios finales. Además de cumplir un rol crítico con operadores importantes como *AT* & *T*, *Bell* Canadá y *Orange*, así como con *OTT* y compañías de medios como Tencent y otros, escuchamos periódicamente de los administradores de pequeños *COP* y *PSI* locales de África y Asia occidental. Como componente clave de *ONAP* y otros marcos de alto nivel, *ODL* está expandiendo cada vez más su presencia global a medida que la automatización de redes se acelera en todo el mundo.

ODL surgió del movimiento *SDN*, con el objetivo de inyectar programabilidad a la red, *ODL* cuenta con más de 1.000 desarrolladores y cuenta con 50 miembros corporativos con soporte para más de 1.000 millones de suscriptores, datos que hablan del apoyo por parte de la comunidad global *SND* para proporcionar mejoras continuas (OpenDaylight Project a Series of LF Projects, 2018).

2.7 Restconf

Según lo descrito en la wiki oficial de OpenDaylight (OpenDaylight, n.d.) *RESTCONF* es un protocolo tipo *REST (representational state transfer),* el cual utiliza el protocolo *HTTP* para acceder a los *DATASTORES* que son contenedores de datos localizados dentro del controlador *ODL*, para acceder

a dichos contendores se utilizan diferentes peticiones *"CRUD" (Create, Read, Update y Delete) HTTP* las más comunes son descritos en la tabla 3:

PETICIÓN *HTTP RESTCONF*	**DESCRIPCIÓN**
POST	Son usados para crear recursos nuevos.
GET	Se usan para leer o recuperar una representación de un recurso dentro de una *URL (Uniform Resource Locator)*, esta petición retorna una representación en formato *XML (eXtensible Markup Language)* o *JSON (JavaScript Objet Notation)* con esta opción solo se puede leer datos no modificar datos.
PUT	La mayoría de veces es utilizado para la actualización de un recurso dentro de *URL* que lo contiene este debe contener todo el cuerpo de los datos en el formato *JSON o XML*.
DELETE	Como su nombre lo indica se usa para eliminar un recurso identificado por una *URL.*
PATCH	Se usa para modificar, esta petición solo necesita los cambios, no los datos del recurso completo, es decir el cuerpo de los datos debe estar en un tipo de lenguaje tipo parche como *JSONPATCH o XMLPATCH.*

Tabla 3. Tipos de peticiones HTTP vía REST. Fuente: Autor

Existen dos tipos de *DATASTORES* o contenedores de datos para ser accesados y poder obtener información o ingresar datos de configuración a los dispositivos de red vía *RESTCONF* dentro del controlador *ODL* descritos en (OpenDaylight, n.d.)

1. **Config DATASTORE:** contiene información de configuración la cual el usuario ha ingresado al *ODL*, es decir allí podremos registrar entradas de flujos, que serán propagadas a los dispositivos del plano de datos, a través de las *Southbound interface* como *OPENFLOW*.

2. **Operational DATASTORE:** Contiene información operacional que ha registrado el sistema en tiempo de ejecución, por ejemplo, el estado de un elemento de la red, la topología o estadísticas de tráfico, solo se permite la lectura de datos en este contenedor.

2.7 Oracle Vm VirtualBox

VirtualBox es un potente producto de virtualización, tanto para uso empresarial como doméstico, es la única solución profesional que está disponible libremente como *software* de código abierto, se están desarrollando activamente actualizaciones con el respaldo de *Oracle* que garantiza la calidad profesional del producto.

Según la información alojada en (Oracle VM VirtualBox, 2018) *VirtualBox* actualmente cuenta con soporte de instalación para diferentes sistemas operativos *(Windows, Linux, Macintosh y Solaris),* permite ejecutar varias máquinas virtuales dentro de un mismo *hardware* y a su vez interactuar con máquinas reales, donde la única limitante son las capacidades de procesador, memoria y disco duro del hardware del equipo anfitrión.

De acuerdo con lo anterior *VirtualBox* será la aplicación que utilizaremos para virtualizar el controlador *OpenDaylight y el Mininet*.

2.8 Wireshark

Wireshark es el analizador de protocolos de red que permite analizar de forma detallada lo que sucede dentro de la red, el desarrollo del *software* prospera gracias a las contribuciones voluntarias de las comunidades de redes en todo el mundo, algunas de las características que presenta *wireshark* se mencionan a continuación, justificando el uso de esta herramienta para nuestro proyecto (Wireshark Foundation, 2018) :

- Inspección profunda de cientos de protocolos de red en todas las capas del modelo *OSI.*

- Es un *software* multiplataforma es decir que se puede ejecutar en *Windows, Linux, Mac OS, Solaris* entre otros.

- Los datos de red capturados se pueden navegar a través de una *GUI* permitiendo una interface gráfica muy fácil de manejar.

- El análisis de protocolos de red se realiza en tiempo real de manera rápida y eficaz.

2.9 Firewall

Un *firewall* o *cortafuegos* es un dispositivo de red que proporciona seguridad a la misma mediante el monitoreo de flujos de tráfico entrantes y salientes, este dispositivo es el encargado de decidir si se permite o se bloquea un tráfico especifico de acuerdo a un conjunto de reglas o políticas de seguridad previamente establecidas.

Estos dispositivos han sido la primera línea de defensa de seguridad de la red durante más de 25 años, establecen un tipo de barrera entre la red interna y redes externas que se consideran no seguras como Internet, existen varios tipos de *firewalls* que pueden estar basados en hardware, software o ambos (Cisco Systems, n.d.).

Firewall Proxy: Es un tipo de firewall que sirve como puerta de enlace de una red hacia otra para una aplicación específica. Pueden proporcionar funcionalidad adicional, como el almacenamiento en memoria caché de contenido, proporciona seguridad ya que evita las conexiones directas desde fuera de la red. Sin embargo, esto puede afectar las capacidades de rendimiento de las aplicaciones.

Statefull inspection firewall: Es considerado hoy en día como un *firewall* "tradicional", permite o bloquea el tráfico según el estado, el puerto y el protocolo. Supervisa toda la actividad desde la apertura de una conexión hasta que se cierra. Las decisiones de filtrado se basan tanto en las reglas definidas por el administrador como en el contexto, es decir al uso de información de conexiones anteriores y paquetes que pertenecen a la misma conexión.

Unified threat management (UTM) firewall: Un dispositivo *UTM* generalmente combina, las funciones de un *firewall* de inspección con estado, con la inspección de intrusión y antivirus. Los *UTM* se enfocan en la simplicidad y la facilidad de uso, permite además que su administración sea en la nube.

Firewall de nueva generación (NGFW): Los *firewalls* han evolucionado más allá del simple filtrado de paquetes y la inspección con estado. La mayoría de las empresas están implementando *firewalls* de última generación para bloquear las amenazas modernas, como *malware* avanzado y ataques en la capa de aplicación de *OSI*. Un firewall de próxima generación contiene las siguientes características:

- Capacidades de firewall estándar como inspección con estado.
- Prevención integrada de intrusos.
- Control para identificar aplicaciones riesgosas para la red.
- Actualizar las rutas para incluir futuras fuentes de información.
- Técnicas para abordar las amenazas de seguridad en evolución.

Threat-focused NGFW: Estos firewalls incluyen todas las capacidades de un *NGFW* tradicional y también proporcionan detección y reparación de amenazas avanzadas. Con un *NGFW* permite lo siguiente:

- Saber qué activos corren más riesgo con conocimiento completo del contexto.
- Reacción rápida ante los ataques con la automatización de seguridad inteligente que establece políticas y fortalece las defensas de forma dinámica.
- Detectar mejor la actividad evasiva o sospechosa con la correlación de eventos de red y punto final.

- Reduce en gran medida el tiempo desde la detección hasta la limpieza con seguridad retrospectiva que supervisa continuamente la actividad y el comportamiento sospechoso incluso después de la inspección inicial.

- Facilite la administración y reduzca la complejidad con políticas unificadas que protegen continuamente la red durante el ataque.

2.10 Estado del arte

En este apartado se exponen algunos documentos y publicaciones que se relacionan con el tema principal del proyecto.

La necesidad de poseer redes más flexibles, escalables e interoperables, ha despertado el interés de hacer uso de *SDN (Software Defined Network),* para mitigar la dependencia de la infraestructura física y llevar ciertas funcionalidades a un entorno virtualizado y administrado por software.

Como se explica en (Diego Kreutz, 2014) en las redes actuales los planos de control y de datos están agrupados y distribuidos a lo largo de la red, lo que conlleva a tener una infraestructura poco escalable, rígida y difícil de gestionar; *SDN* se presenta como un paradigma emergente que promete solucionar estas dificultades desligando la integración de estos planos, separando la red lógica de control de los conmutadores y enrutadores, centralizando el plano de control y permitiendo la capacidad de programar la red.

En esa medida, la seguridad y el control de tráfico se han convertido en temas de gran interés, razón por la cual se ha optado por utilizar diferentes mecanismos que permitan proteger los servicios disponibles en la red, así como los dispositivos involucrados en la prestación de éstos. Por ejemplo, en (Arins, 2015) se propone un *firewall* como servicio en las redes de los *ISP (Internet Service Provider)* que emplee políticas de red con el fin de descartar tráfico indeseados en los enrutadores de borde de los *ISP*. Para ello se crea un entorno *SDN* bajo el protocolo *OpenFlow* y usando *switches OpenFlow* y un controlador *ONOX*, a partir de lo cual se obtiene una *API (Application Programming Interface)* remota que permite mitigar ataques *DDoS (Distributed Denial of Service)*. Así mismo, en (Wajdy M. Othman, 2017) se realizó una implementación de algunas funcionalidades de firewall las cuales permitían definir políticas de red en el controlador *SDN POX* para luego ser aplicadas en los dispositivos del plano de datos, para llevar a cabo dicho proyecto utilizaron varias herramientas de software opensource, como el caso de *Mininet* y *VirtualBox* los cuales permitieron la creación de un entorno de red *SDN* virtualizado y para medir el rendimiento del firewall emplearon *Wireshark* e *Iperf*, como resultado obtuvieron un firewall *SDN* capaz de detectar tráfico y aplicar políticas de red en las capas 2/3/4 del modelo de referencia *OSI* y trabajar en conjunto con el controlador *SDN POX* y un *switch* virtual *OpenvSwitch*.

3. METODOLOGÍA

En este capítulo se describe detalladamente, como han sido desarrollados cada uno de los objetivos específicos definidos en el capítulo 1 para así construir el objetivo general y dar a cabalidad el proyecto final; además se exponen los recursos tecnológicos de hardware y de software utilizados a lo largo del desarrollo con el fin de identificar cuáles fueron los requisitos necesarios para la elaboración del proyecto.

El objetivo general de este proyecto es obtener un prototipo de firewall para redes definidas por software que permita traducir políticas de seguridad, diseñadas en un lenguaje intuitivo para los usuarios y operadores de redes, en acciones de detección, mitigación y control de tráfico en dispositivos de red programables.

A continuación, se expone la metodología propuesta para cumplir con cada uno de los objetivos específicos y poder construir el objetivo general y dar cabalidad al proyecto final.

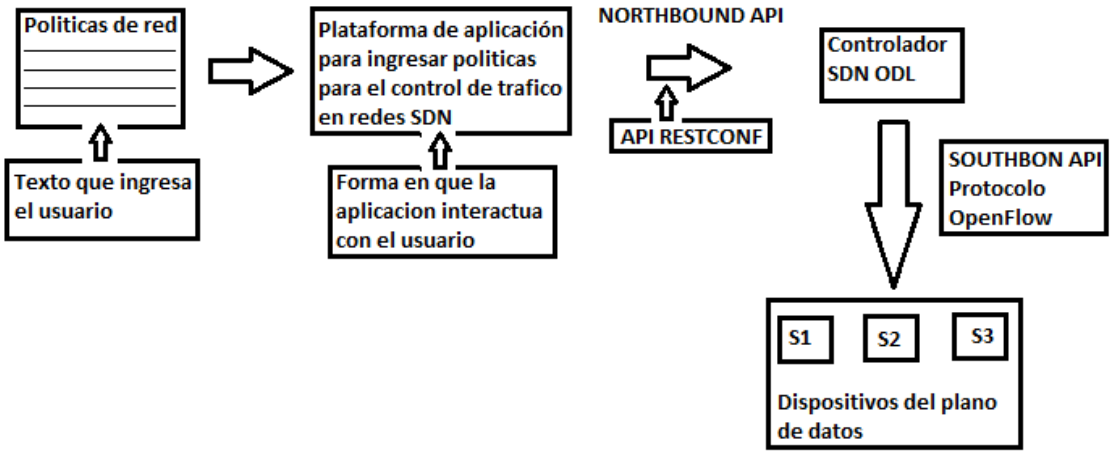

Ilustración 5. Diagrama con los componentes que conforman el objetivo del proyecto. Fuente: Autor.

3.1 Investigación sobre las generalidades de SDN y aplicaciones de firewalls en entornos SDN

En esta primera fase se realizó una investigación muy detallada donde se consultaron diferentes fuentes de información como las bases de datos suscritas del *ITM, wikis* y sitios *web* oficiales con el objetivo de obtener los conocimientos necesarios sobre las generalidades de *SDN*, las herramientas tecnológicas, protocolos implicados entre otros, con el fin de facilitar su comprensión y a su vez definir la metodología y los recursos tecnológicos necesarios para la ejecución del proyecto, toda la información que se consideró pertinente y de gran vitalidad residen en el capítulo 2 dentro del marco teórico, estado del arte y las citas bibliográficas.

3.2 Creación del entorno de simulación controlado

Como se mencionó anteriormente en el capítulo 1 el proyecto se desarrollará en un ambiente de simulación utilizando varias herramientas tecnológicas que son de vital importancia para la ejecución de este proyecto.

3.2.1 Hardware empleado

Para el desarrollo de esta etapa se utilizó un computador portátil marca Toshiba, modelo *Satellite L655,* con un procesador *Intel® Core i5 CPU M 480 @ 2.67 GHZ* con *8 GB* de memoria *RAM y* un sistema operativo *Windows 7* de *64 bits* y procesador *x64.* En el ordenador se instalaron diferentes *softwares open source* las cuales no suponen ningún costo monetario para su implementación además de contar con varias *wikis* y sitios *web* con tutoriales e información de gran utilidad, esta fue la principal razón por la cual se optó por utilizar estas herramientas.

3.2.2 Mininet

Para crear la topología virtual dentro del Mininet inicialmente se descargó de la página www.mininet.org, la herramienta para simular redes *SDN*, la versión 2.2.2 de *Mininet* ya que es la versión más actualizada y más estable a la fecha, posteriormente se instaló en el *hypervisor Oracle VM VirtualBox* que se descargó de la página www.virtualbox.org, a partir de ello se iniciará el diseño de la topología de red *SDN* virtual (Controlador *SDN OpenDaylight*, Conmutadores *OpenvSwitch* y host finales).

A la máquina virtual *Mininet-VM* se le asignó 1.024 *MB* de memoria *RAM*, una *CPU* y 8 *GB* de disco duro, con estos recursos basta para realizar la topología de red *SDN* que necesitamos para el proyecto, también se le asignaron dos adaptadores de red virtuales, una de ellas debe ser una interface *NAT* que se usó para proveer acceso a internet y descargar los paquetes y actualizaciones que se necesiten dentro de la máquina virtual, la otra debe ser una interface llamada "*VirtualBox Host-Only Ethernet Adapter"* la cual conecta la máquina virtual con la maquina real y el controlador *ODL* además podemos usar conexiones *SSH (Secure Shell)* para administrar el Mininet de una mejor forma, pero su función principal es permitir la interconexión con el controlador *ODL*.

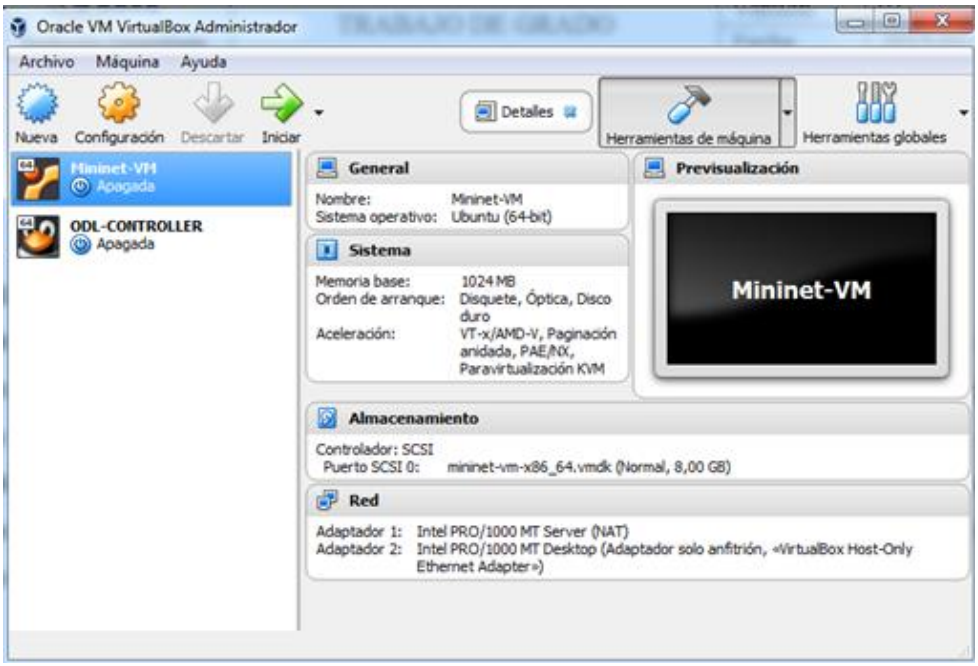

Ilustración 6. Características de instalación Mininet dentro del VirtualBox. Fuente: Autor.

3.2.3 Creación de la topología en Mininet.

Mininet es una herramienta que ofrece varias formas para crear topologías, una de ellas es mediante la *CLI* empleando comandos, otra mediante una *GUI Miniedit* y por ultimo mediante la elaboración de un *script* en el lenguaje de programación *Python*, cabe destacar que *Mininet* es basado en ese lenguaje de programación por lo tanto en ese lenguaje se pueden crear *scripts* basados en *Python* y el *Mininet* lo ejecutará sin ningún inconveniente , a continuación se expone la topología que hemos creado en *Mininet.*

En este punto se diseñó la topología a partir de la interface de línea de comando que ofrece el *Mininet*, para crear la topología dentro del *Mininet,* la cual consiste en 3 *switches Open vSwitch (Openflow:1, OpenFlow:2 y OpenFlow:3)* cada uno conectado entre sí, además como se mencionó anteriormente vienen con soporte para el protocolo *OpenFlow* que es utilizado como *API Southbound* para permitir la comunicación con el controlador *ODL,* también se crearon 3 host *(h1,*

h2 y h3) y además se indicó que se utiliza un controlador remoto *OpenDaylight,* en la siguiente imagen se muestra como fue creada la topología.

```
mininet@mininet-vm:~$ sudo mn --topo=linear,3 --mac --controller=remote,ip=192.1
68.56.104,port=6633 --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
Unable to contact the remote controller at 192.168.56.104:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

Ilustración 7. Creación de la topología dentro del Mininet. Fuente: Autor

En la anterior imagen se muestra los comandos utilizados para crear la topología a continuación se describe los componentes que lo conforman:

- ***--topo=linear,3:*** Indica que la topología que se pretende crear consiste de 3 *switches (S1, S2 y S3)* conectados entre sí y que cada uno tiene conectado un host *(h1, h2 y h3)* respectivamente.
- ***--mac:*** *Mininet* por defecto crea direcciones *MAC* de forma aleatoria con este parámetro las podemos asignar de forma organizada a cada una de los *hosts* de la topología, a modo de ejemplo h1 tiene la dirección *IP* 10.0.0.1 entonces su dirección *MAC* será 00:00:00:00:00:01.
- ***--controller=remote, ip=<ip del controlador>, port=<número de puerto>:*** Esto representa que el controlador será remoto, se indica la dirección IP del controlador y el número de puerto de escucha del controlador.
- ***--switch ovsk, protocols=OpenFlow13=*** Indica que los *switches* dentro de la topología serán *Open vSwitch* y que el protocolo que se integrará es *OpenFlow* en su versión 1,3.

A continuación, se presenta una tabla donde se ilustra los parámetros de red de los dispositivos involucrados en la topología *SDN.*

DISPOSITIVO	DIRECCIÓN IP	DIRECCIÓN MAC
OpenFlow:1	192.168.56.102 /24	9A:63:11:DA:8B:42
OpenFlow:2	192.168.56.102 /24	32:96:19:A5:DD:3A
OpenFlow:3	192.168.56.102 /24	C6:52:67:19:7A:7A
H1	10.0.0.1/24	00:00:00:00:00:01
H2	10.0.0.2/24	00:00:00:00:00:02
H3	10.0.0.3/24	00:00:00:00:00:03
ODL	192.168.56.104/24	----

Tabla 4. Características de los dispositivos involucrados en la topología. Fuente: Autor.

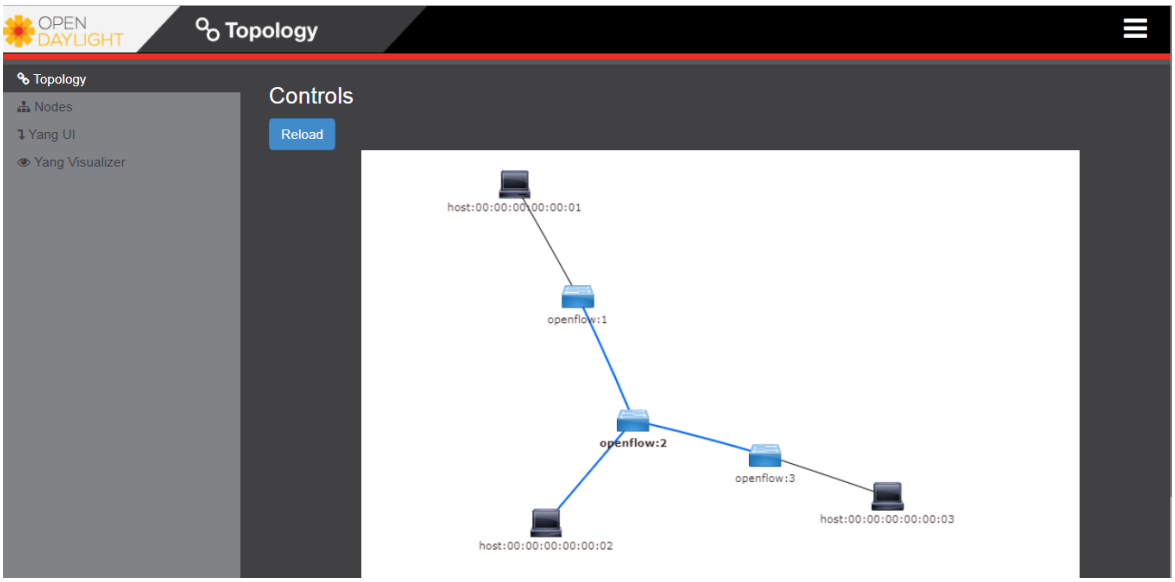

Ilustración 8. Topología vista en la interface Web ODL DLUX. Fuente: Autor.

En la anterior imagen se ilustra mediante la interface *web* del controlador *ODL DLUX* la topología creada en el *Mininet.*

3.2.4 INSTALACIÓN DEL CONTROLADOR SDN OPENDAYLIGHT

Para construir el controlador *ODL,* se descargó la imagen *ISO Ubuntu Server 16.04.4 LTS* de 64 *bits,* desde el sitio web https://www.ubuntu.com/download/server , se utilizó esta versión porque es la versión más actualizada y estable de *Ubuntu server* y cuenta con soporte por 5 años, además incluye la última versión de Open vSwitch, 2.5.0. que también es una versión LTS de Open vSwitch, posteriormente se instaló en el *hypervisor Oracle VM VirtualBox.*

A la máquina virtual *ODL-CONTROLLER* se le asignó *2 CPU, 2.048 MB* de memoria *RAM* y *12 GB* de disco duro, ya que estos son los recursos mínimos para tener un óptimo rendimiento del controlador *ODL*, al igual que con la máquina virtual *Mininet-VM,* se le asignaron 2 adaptadores de red, una interface NAT para proveer el acceso a internet al controlador y descargar los ficheros y actualizaciones necesarias, y también un adaptador *VirtualBox Host-Only Ethernet Adapter* para realizar conexiones *SSH* y establecer un canal entre el controlador y el *Mininet.*

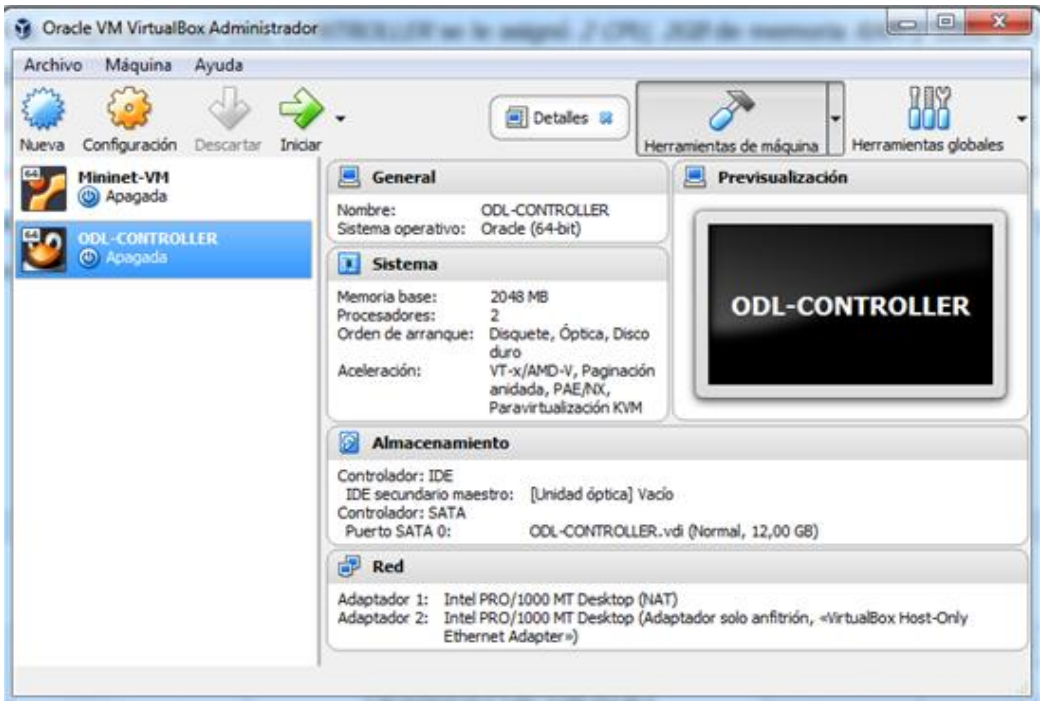

Ilustración 9. Características del controlador ODL dentro del VirtualBox. Fuente: Autor.

3.3 Proveer una interfaz que permita establecer un protocolo de comunicación entre la capa de aplicación y la capa de control (Northbound API).

Como se mencionó en el capítulo 2, las *Northbound API* son utilizadas por las aplicaciones de *SDN* las cuales usan el controlador *ODL* para recopilar o modificar información de los dispositivos de red y luego ejecutar las acciones mediante las *API Southbound.* También cabe recordar que inicialmente el controlador no tiene instalado ningún "micro servicio" previamente.

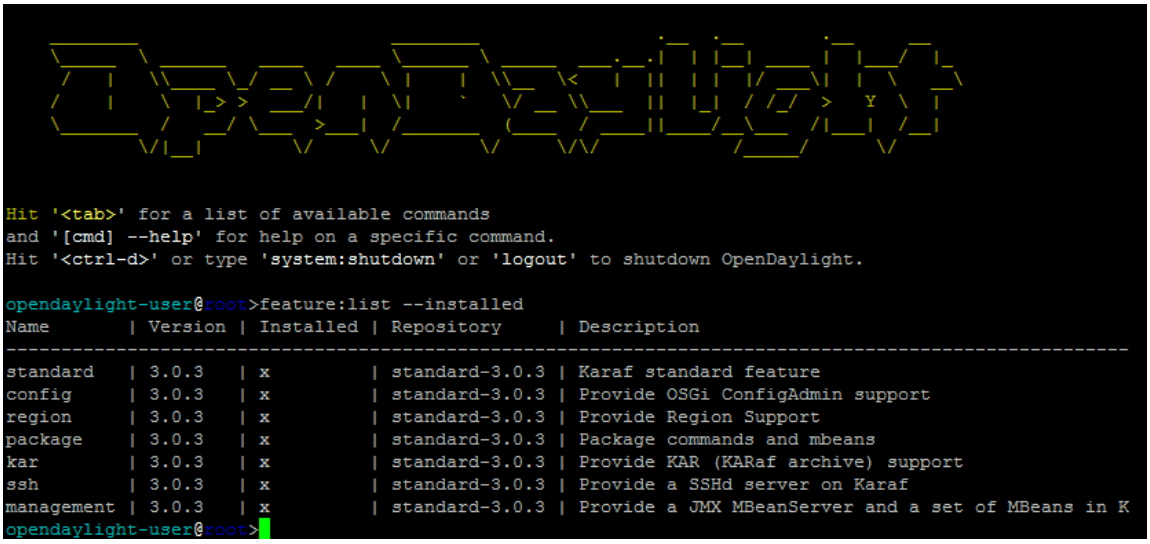

Ilustración 10. Características por defecto del controlador. Fuente: Autor.

OpenDaylight hace uso de dos caminos para operar con el controlador:

1. **Bundles:** Son paquetes que se instalan dentro de *Karaf* y son ejecutados de manera local, al trabajar con esta opción nos permite la instalación de ***flujos reactivos*** (Aquellos que se instalan al llegar un paquete al switch y cumplen con unas características predefinidas) y también permite trabajar con ***flujos proactivos*** (Que son flujos instalados con antelación a la llegada de cualquier paquete anticipadamente).
2. **Instrucciones REST:** Son peticiones tipo *REST (Representational State Transfer),* que utiliza el protocolo *HTTP (Hypertext Transfer Protocol)* para establecer comunicación con el controlador *ODL.*

3.3.1 Instalación de la *Northbound API RESTCONF*

En esta etapa se instalaron los "micro servicios" que se utilizaron para establecer la comunicación entre la capa de aplicación y el controlador es decir la *Northbound API, RESTCONF es la Northbound API* que se usó para dicho propósito, como se mencionó anteriormente RESTCONF es la *Northbound* utilizada para acceder a los *Datastores o* contenedores de datos alojados en el controlador ODL. Algunas características de la Northbound API se presentan a continuación:

- Escucha peticiones a través del puerto 81:81 vía HTTP.
- Soporta las peticiones *HTTP GET, PUT, POST y DELETE* donde las peticiones y respuestas pueden generarse en formatos XML o JSON.
- Hace uso del protocolo *NETCONF* que es empleado para gestionar y configurar dispositivos de red.

Para instalar este micro servicio basta con escribir el siguiente comando en la *CLI* del controlador *ODL*.

```
opendaylight-user@root>feature:install odl-restconf
```

Ilustración 11. Instalación de la Northbound API dentro del controlador. Fuente: Autor.

3.3.2 Envío peticiones al controlador vía *RESTCONF*

Existen diferentes alternativas para enviar peticiones al *ODL* a través de *RESTCONF,* una de ellas es por medio de *softwares* como *POSTMAN, HTTP REQUESTER o RESTED*, otra forma es utilizando una interfaz gráfica en el controlador *ODL, DELUX (open**D**ay**L**igth **U**ser **E**xperience)* que es un micro servicio que se instaló dentro de *ODL* y es básicamente una interfaz web que nos permite obtener información de la topología, flujos, estadísticas, ingresar y recibir peticiones *HTTP* vía RESTCONF hacia los *Datastores* del controlador para configurar dispositivos del plano de datos. Estas herramientas nos proporcionan automatizar procesos, almacenar peticiones, personalizar encabezados, etc.

A continuación, se muestra la instalación el micro servicio Opendaylight User Experience y la interface web que proporciona dicho servicio.

```
opendaylight-user@root>feature:install odl-dlux-all
```

Ilustración 12. Instalación de la interfaz web DLUX dentro del ODL. Fuente: Autor.

Una vez instalado el micro servicio *ODL DLUX* para ingresar a la interface web es necesario digitar en el navegador web http://<IP_DEL_CONTROLADOR>:8181/index.html#/login, y a continuación, se procede a ingresar el usuario y la contraseña de autenticación que para ambos casos es "*admin*".

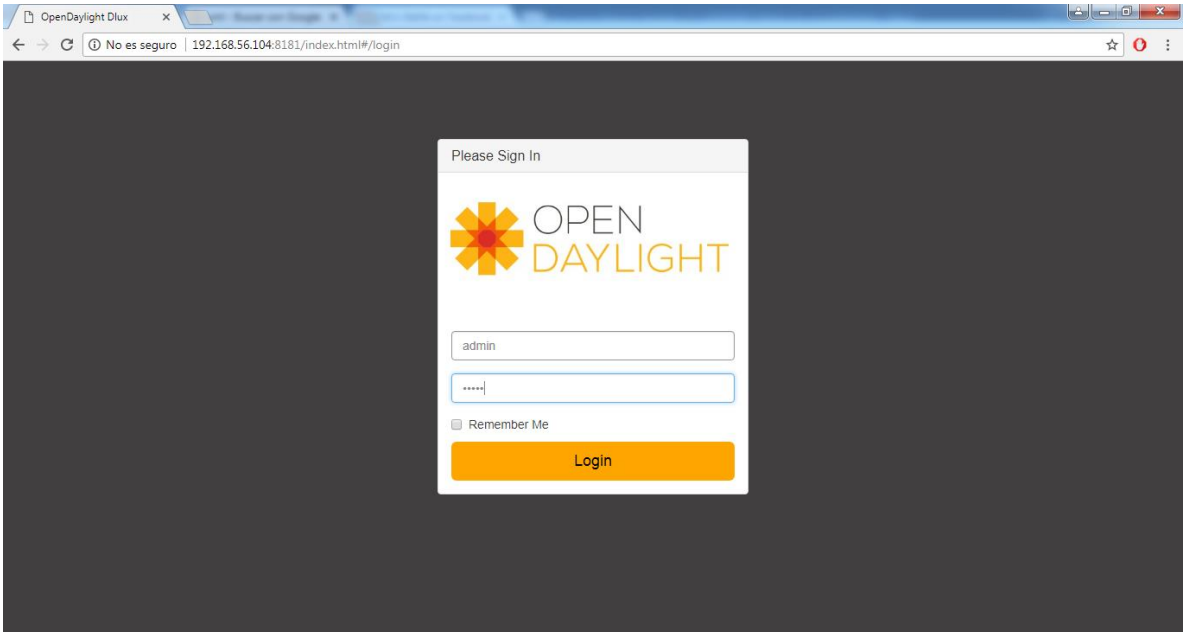

Ilustración 13. Interfaz Web ODL DLUX. Fuente: Autor.

En nuestro caso se utilizó el software *POSTMAN* por ser una herramienta fácil de emplear y ser una aplicación *opensource*, se descargó e instaló el software desde el sitio oficial https://www.getpostman.com/apps. A continuación, se presenta la interfaz gráfica de la aplicación donde se realizó una prueba utilizando *RESTCONF* como *Northbound API* donde se hace una petición *HTTP GET* solicitando información de la topología al contenedor de datos "*DATASTORE*" *operational* dentro del *ODL.*

3.3.3 Pruebas de peticiones GET y PUT vía RESTCONF utilizando la herramienta POSTMAN.

A continuación, se realizaron las siguientes pruebas con el fin de verificar que los datos enviados por el usuario son recibidos por el controlador *ODL* y este a su vez los envía a los dispositivos del plano de datos dentro de la topología creada en el *Mininet*, y a su vez las entradas de flujo son aplicadas correctamente dentro de los *switches Open vSwitch.*

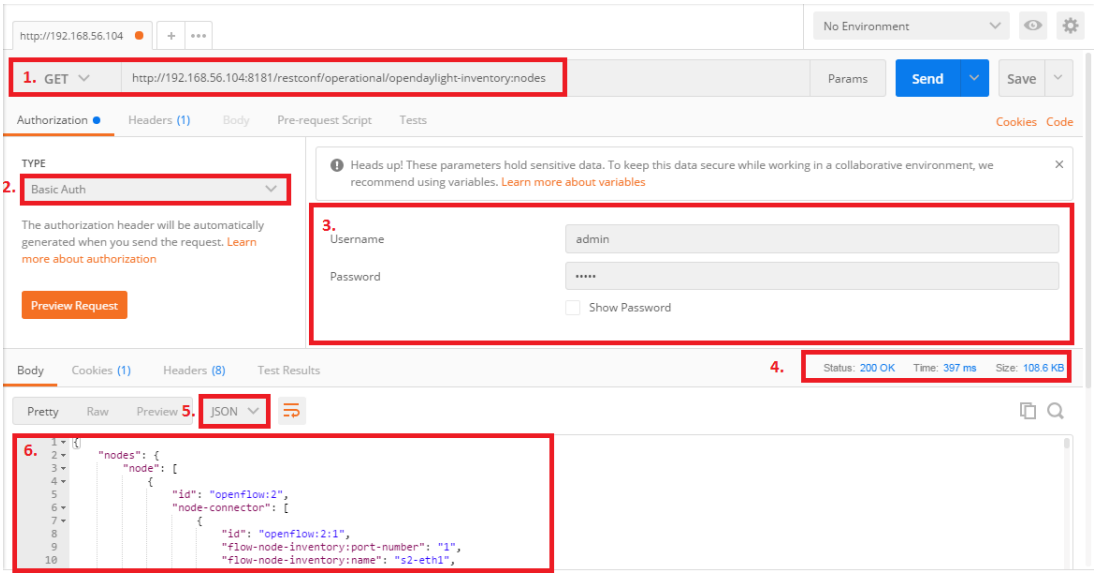

Ilustración 14. Prueba GET en Postman. Fuente: Autor.

La anterior imagen corresponde a una prueba donde se envió una solicitud HTTP GET con el fin de obtener una descripción de la topología de la red, los pasos que se realizaron fueron los siguientes:

1. Envío de una solicitud *HTTP GET* a la *URL* http://192.168.56.104:8181/restconf/operational/opendaylight-inventory:nodes en la descripción del enlace se puede observar que se digito la dirección IP del controlador *ODL*, seguido del número de puerto *8181* que como se mencionó antes es el puerto habilitado para la *API RESTCONF*, seguido del Datastore operational que es el contenedor de solo lectura y se solicitó información del inventario de los nodos (Dispositivos del plano de datos alojados en el *Mininet*) que controla el *ODL*.

2. Se seleccionó el tipo de autenticación *Basic-Auth*.

3. Como se mencionó anteriormente la interface *web DLUX* autentica con "*admin*" para el *login* y el *password*.

4. Si la solicitud *HTTP* fue enviada y recibida con éxito aparece un mensaje con el código *200 ok* y el tiempo que duró la solicitud, y también el tamaño del archivo de respuesta.

5. Se debe elegir el tipo de archivo que se quiere obtener como respuesta para este caso se eligió en formato *JSON.*

6. Se puede ver el archivo de la respuesta por parte del *ODL* en formato *JSON* y en él se aprecia la información de la topología. En el apéndice B se puede apreciar con más claridad.

A continuación, se presenta una imagen donde se ilustra una petición *HTTP PUT* con la herramienta *POSTMAN*, con el objetivo de ingresar una entrada de flujo en formato *JSON* al controlador *ODL* y verificar que el controlador si está entendiendo dicho flujo y a su vez comprobar dentro del Mininet que los dispositivos del plano de datos están instalando dicho flujo.

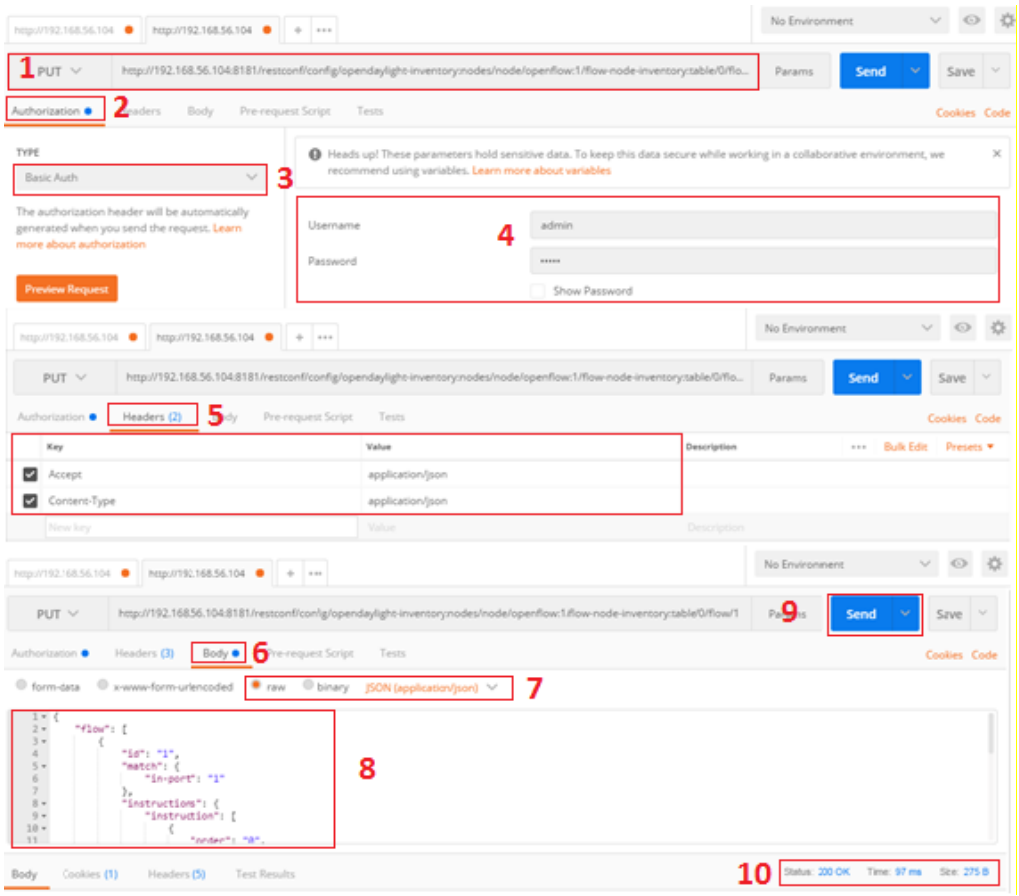

Ilustración 15. Solicitud HTTP PUT al controlador vía POSTMAN. Fuente: Autor.

A continuación, se describen los pasos realizados en la prueba *HTTP PUT*:

1. Envío de una petición *HTTP PUT* a la siguiente *URL* *http://192.168.56.104:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/flow-node-inventory:table/0/flow/1* , en el enlace se puede aprecia que se colocó la dirección *IP* del controlador y el número de puerto que corresponde al puerto para *REST API,* también se observa que la petición *HTTP PUT* se realiza mediante el micro servicio *RESTCONF* y se aplicara sobre el *Datastore Config* que como se mencionó en el capítulo 2, es donde se realizan las configuraciones, se selecciona el nodo al que queremos aplicar en este caso openflow:1, además se debe indicar los números de tabla flujo y de la entrada de flujo que para este caso es la tabla 0 y el flujo 1.

2. Seleccionamos la pestaña *Authorization.*

3. Seleccionamos el tipo de autenticación en este caso es *Basic Auth.*

4. Como se mencionó anteriormente la interface *web DLUX* autentica con "*admin*" para el *login* y el *password*.

5. En la pestaña *Headres* se indica los encabezados *Accept* y *Content-Type* lo que indica el tipo de contenido de la petición *HTTP PUT* para este caso el contenido esta en formato JSON.

6. Seleccionamos la pestaña *Body*.

7. Elegimos el tipo de formato que contiene el cuerpo de la petición *HTTP PUT* en este caso es JSON.

8. Ingresamos el cuerpo de la petición; en el apéndice C se puede apreciar con mayor claridad.

9. Seleccionamos enviar.

10. Si la solicitud *HTTP* PUT fue enviada y recibida con éxito aparece un mensaje con el código *200 ok,* el tiempo que duró la solicitud y también el tamaño del archivo de respuesta.

3.4 Desarrollar un mecanismo de capa de control que permita traducir políticas de seguridad en reglas de flujo para los dispositivos en la capa de datos.

En este punto se utilizó el protocolo abierto *OpenFlow*, que permite establecer la comunicación entre el controlador *ODL* y los dispositivos en el plano de datos *Open vSwitch*, para ello se instaló el siguiente micro servicio *OpenFlowplugin,* en la siguiente imagen se ilustra cómo fue instalado.

```
opendaylight-user@root>feature:install odl-openflowplugin-all
```

Ilustración 16. Instalación de Southbound API dentro del controlador. Fuente: Autor.

3.5 Diseño de una aplicación de red que permita instruir políticas de seguridad a un controlador SDN usando un lenguaje e interfaz intuitivos para usuarios y operadores de red.

En este etapa se definió la forma en que los usuarios y operarios de la red van a interactuar con la aplicación de red para instruir las políticas de seguridad al controlador, para ello se definió que dicha interacción sea de una forma muy intuitiva y fácil para cualquier usuario de redes, nuestra topología creada en el *Mininet* consta de *3 switches S1, S2 y S3*, tres *host*s finales *H1, H2 y H3,* cada uno conectado a un *switch* y por ultimo un controlador *ODL* remoto, todos los dispositivos pertenecen a la misma red *LAN,* de esta topología se profundizó en la sección 3,2 de este mismo capítulo, de acuerdo a esta topología fue diseñada la forma en que el usuario interactúa con nuestra aplicación la cual consiste de los siguientes pasos:

I. Inicialmente la aplicación le pregunta al usuario a qué switch quiere aplicarle la política “Ingrese el número del switch correspondiente al que desea aplicar la política”.

Opciones:

1. OpenFlow:1

2. OpenFlow:2

3. OpenFlow:3

II. Definido el switch al que el usuario quiere efectuar la política de red la aplicación le pregunta a el usuario que tipo de política desea aplicar: "Ingrese el número correspondiente al tipo de política que desea aplicar".

Opciones:

1. Reenviar.
2. Bloquear.
3. Permitir.

III. Nuestra plataforma para aplicar políticas de seguridad es capaz de interactuar en las capas 2 y 3 del modelo de referencia *OSI,* es decir el usuario podrá elegir si filtra por dirección MAC por dirección IP o por número de puerto o una combinación de ellas o las tres a la vez, definido esto la aplicación le pregunta al usuario: "Ingrese el número correspondiente al parámetro por el cual desea filtrar el tráfico".

Opciones:

1. MAC origen.
2. MAC destino.
3. IP origen.
4. IP destino.

IV. Adicionalmente se le preguntara al usuario si desea ingresar otro parámetro para filtrar es decir se le pregunta: "Desea ingresar otro parámetro".

Opciones:

1. SI.
2. NO.

En caso tal que el usuario elija la opción 1 llevará a la aplicación de nuevo al paso ***III***, de lo contrario seguirá con el paso ***V.***

V. Finalmente, todas las opciones que el usuario respondió dentro de la aplicación serán almacenadas y mostradas en la *CLI* para que el usuario las observe y posteriormente estos datos serán enviados al controlador *ODL*.

A continuación, se presenta el diagrama de flujo donde se explica de una forma más clara cómo el usuario va interactuar con la aplicación *CLI.*

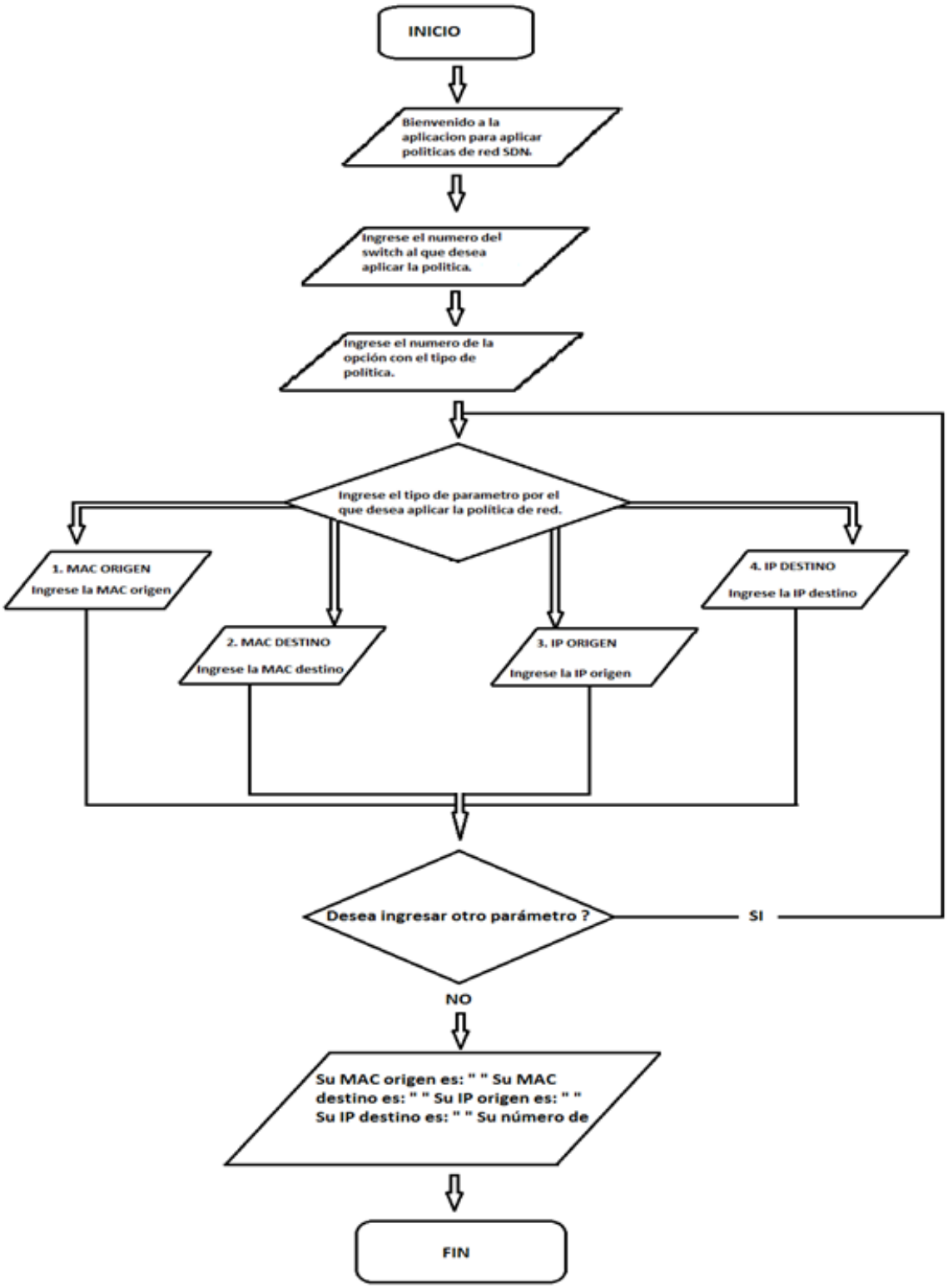

Ilustración 17. Diagrama de flujo que describe el funcionamiento de la aplicación. Fuente: Autor.

Tomando como referencia el anterior diagrama de flujo y los pasos mencionados, procedemos a construir el programa para la aplicación, para ello se descargó e instalo el *software NetBeans IDE* 8.2 desde la página oficial http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html el cual sirve como una plataforma *IDE (Integrated Development Environment)* para desarrollar programas en el lenguaje de programación *JAVA,* se eligió este *software* ya que es libre y fácil de manejar, posteriormente tomando como referencia el diagrama de flujo ilustrado en la pasamos escribir el código en *JAVA* para el desarrollo de la aplicación, se eligió este lenguaje de programación ya que el controlador *ODL* acepta este lenguaje para las *API* en las *Northbound Interface.* En el apéndice A se deja expuesto el código de programación para la aplicación, a continuación, se muestra un ejemplo gráfico de dicha aplicación funcionando.

4. RESULTADOS Y DISCUSIÓN

En esta sección contiene todos los resultados obtenidos de manera clara y concisa, donde se explica cada una de las pruebas realizadas dentro de un entorno de simulación virtual y controlada siguiendo la metodología empleada en el capítulo 3.

Previamente se ejecuta conjuntamente la topología ilustrada en la figura 8 dentro del *Mininet* y se inicia el controlador *ODL*, luego se inicia la aplicación para ingresar las políticas de seguridad dentro de la red *SDN,* a continuación, se ilustra como la aplicación interactúa con el usuario de forma intuitiva para aplicar dichas políticas según el diagrama de flujo expresado en la figura 17.

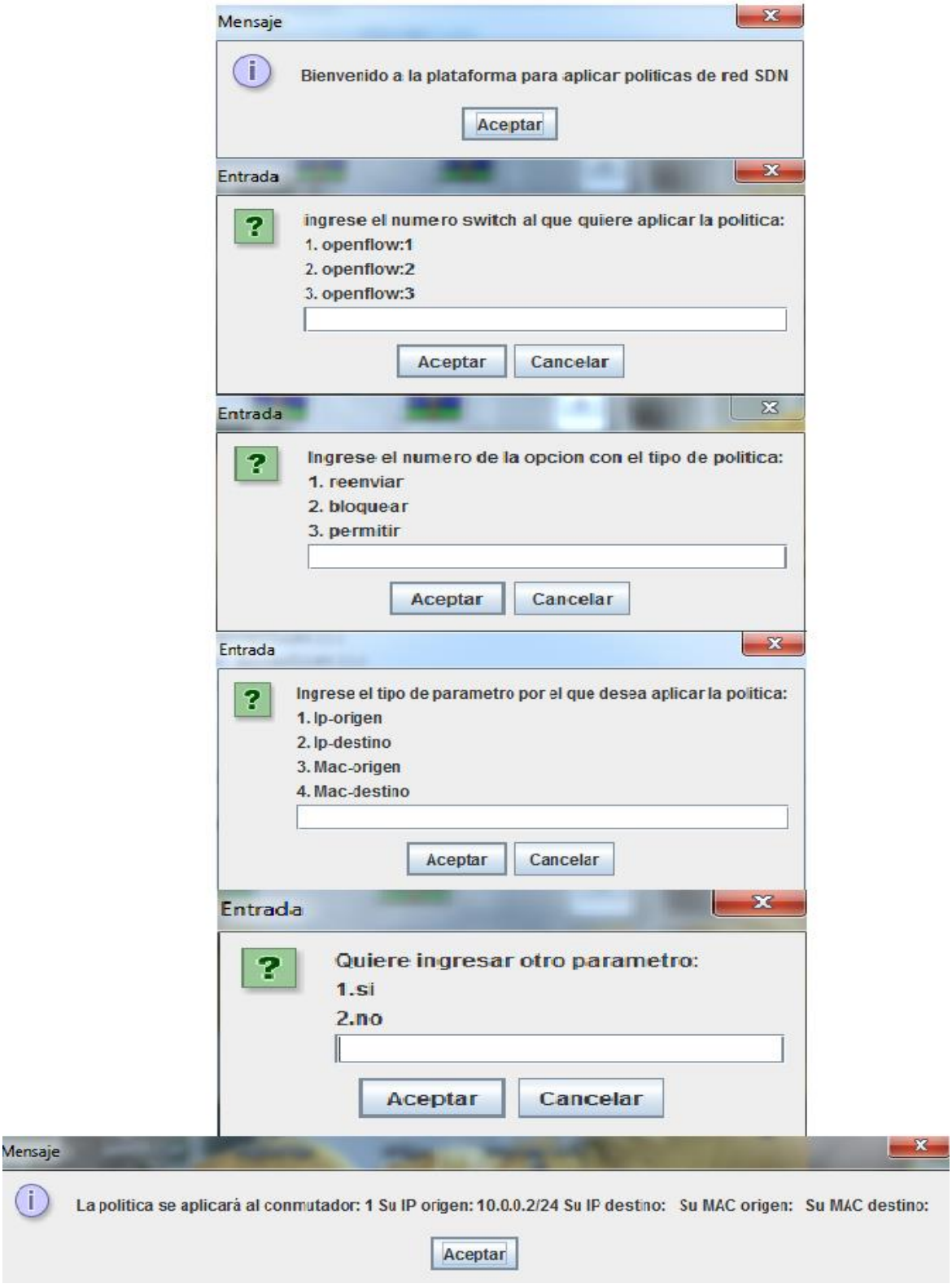

Ilustración 18. Plataforma para aplicar políticas de seguridad en redes SDN. Fuente: Autor.

4.1 Reenvío de tráfico

A continuación, se muestra de forma clara los resultados obtenidos luego de aplicar políticas de reenvió dentro de la topología.

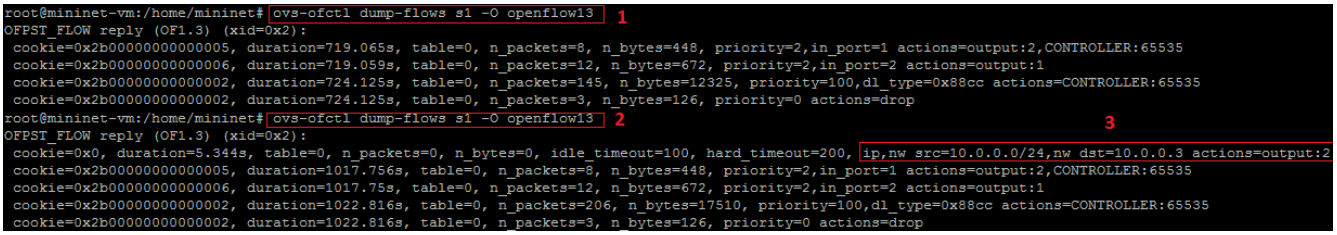

Ilustración 19. Reenvío de tráfico 1. Fuente: Autor.

Para aplicar dicha política partimos del hecho de que el usuario quiere aplicar una política al *switch* 1 para reenviar el tráfico proveniente de una red cuyo destino es un host en específico, porque considera que este tráfico es malicioso y lo reenvía a un puerto en este caso el puerto dos donde está conectado un *firewall* o un *Deep Packet Inspector* para analizar más detenidamente este tráfico

1. Inicialmente se verifica que el *switch 1* no tiene ninguna política aplicada a parte de las básicas de conmutación.
2. Luego de ingresar la política a través de la aplicación y nuevamente se verifica la tabla de flujo del switch 1.
3. Se observa que hay una nueva entrada de flujo que indica cuando llegue un paquete de la red 10.0.0.0 /24 con el destino 10.0.0.3 la acción correspondiente es reenviar ese tráfico al puerto 2.

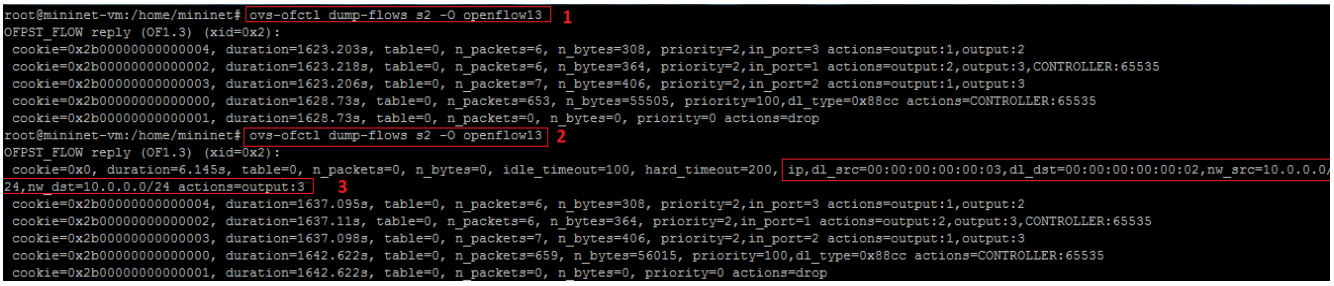

Ilustración 20. Reenvío de tráfico 2. Fuente: Autor.

4.2 Bloqueo de tráfico

A continuación, se presentan los resultados obtenidos luego de aplicar políticas de seguridad para bloquear tráfico.

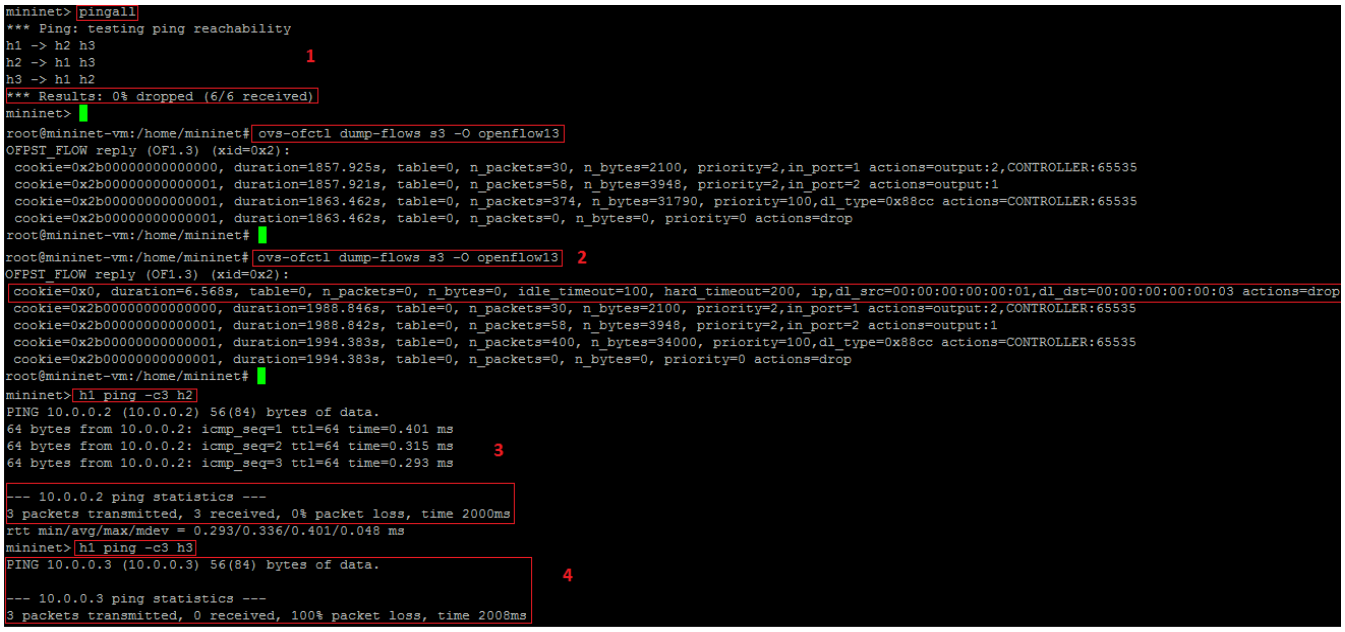

Ilustración 21. Bloqueo de tráfico 1. Fuente: Autor.

El objetivo de aplicar esta política es bloquear el tráfico en la capa 2 del modelo *OSI,* específicamente las direcciones *MAC* origen y destino que corresponden a los host *h1* y *h3* según la tabla 4.

1. Inicialmente verificamos que tenemos conectividad de extremo a extremo con todos los hosts involucrados dentro de la topología para ello se ejecuta el comando *pingall* en el *Mininet,* en la imagen se puede observar que todos los paquetes fueron entregados de forma exitosa.
2. Luego se verificó que luego de ingresar los datos dentro de la aplicación, el controlador *ODL* aplicó la política en el switch 3 que indica que los paquetes que ingresen con la dirección *MAC* de origen 00:00:00:00:00:01 cuyo destino sea la *MAC* de destino

00:00:00:00:00:03, la acción correspondiente que tomara el *switch* es descartar esos paquetes.

3. Nuevamente se prueba conexión entre el host h1 y h2 mediante el comando *ping* y se observa que la comunicación es exitosa ya que la entrada de flujo aplica para la comunicación entre h1 y h3.
4. Se verifica que al realizar comunicación entre los host h1 y h3, esta falló, por consiguiente, se concluye que la entrada de flujo fue aplicada de manera exitosa.

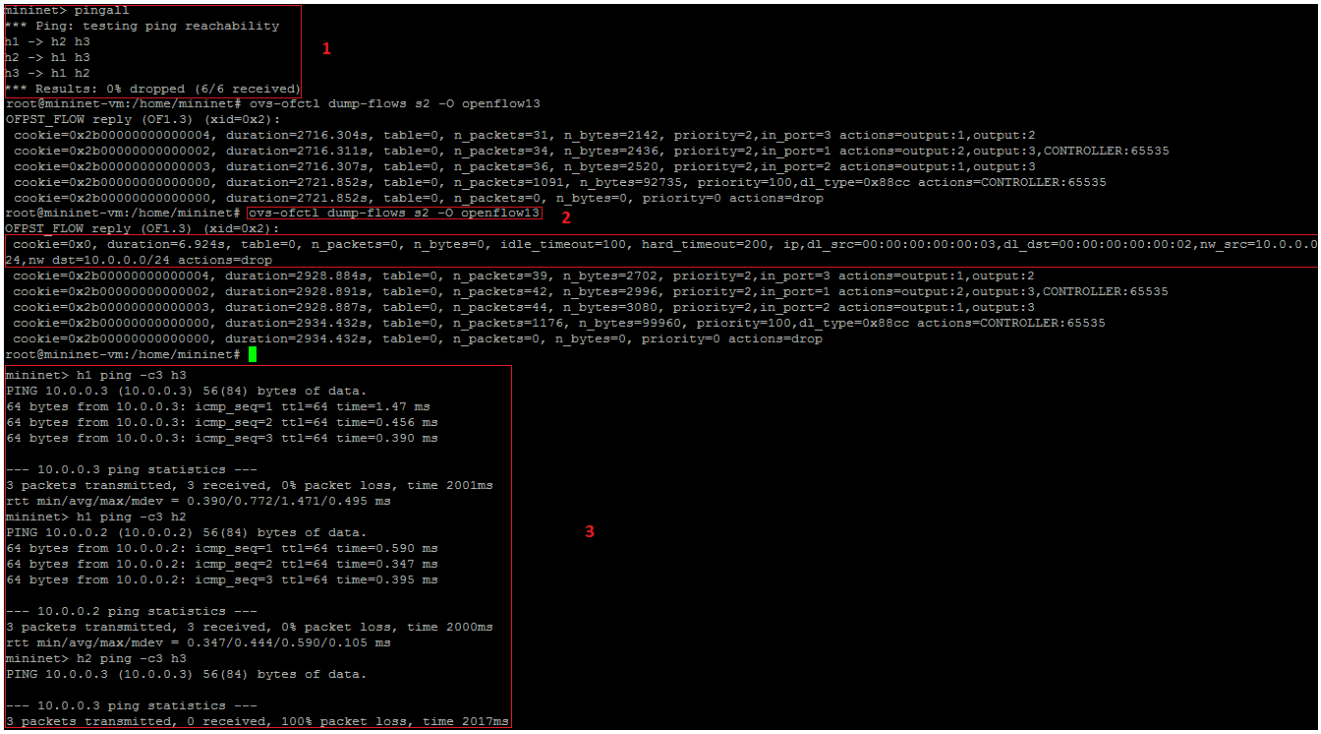

Ilustración 22. Bloqueo de tráfico 2. Fuente: Autor.

La anterior imagen corresponde a la aplicación de una política de seguridad un poco más ambiciosa ya que se filtra tráfico en las capas 2 y 3 del modelo *OSI*, específicamente las direcciones de *h2* y *h3* de acuerdo a la tabla 4.

1. Primero se verifica que no hay ninguna política aplicada en el *switch* 2, por consiguiente, cuando se ejecuta el comando *pingall* en el *Mininet* todos los paquetes fueron enviados y recibidos con éxito.

2. Luego de ingresar los datos en la aplicación, se comprueba que la política fue almacenada en la tabla 0 dentro del *switch* 2.
3. Para finalizar se prueba mediante el comando *ping* la conexión entre *h1 y h2, h1 y h3* la cual muestra que es exitosa pero cuando se hizo para *h2 y h3* los paquetes fueron rechazados esto comprueba que la política funciona de forma correcta.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

5.1 Conclusiones

Al finalizar este proyecto se puede concluir que la plataforma para aplicar políticas de seguridad en redes definidas por software funciona de forma correcta en un entorno virtualizado y controlado *SDN,* y que aplicar dichas políticas se puede lograr de forma intuitiva y fácil para los operarios y administradores de redes definidas por software, permitiendo una gestión mucho más sencilla que en las redes tradicionales.

Al finalizar el proyecto se concluye que trabajar con redes *SDN* es necesario la inclusión de personal capacitado en varias áreas de las *TIC* como son profesionales en redes e ingenieros de sistemas con capacidades para programar en diferentes lenguajes, este último represento un gran reto durante el desarrollo, debido a las falencias que presentamos los estudiantes del programa académico Ingeniería de Telecomunicaciones del ITM.

A forma de contraste entre las redes *SDN y* las redes tradicionales, se concluye que es posible diseñar nuestras propias aplicaciones de red de una forma mucho más flexible ya que no se depende de los fabricantes y las funciones que ellos imponen en sus equipos de redes.

A partir de los resultados obtenidos se puede concluir que la plataforma para aplicar políticas de seguridad en redes definidas por software, permite aplicar políticas de seguridad bajo los criterios de permitir, bloquear y reenviar tráfico dentro de las capas 2 y 3 del modelo de referencia *OSI* presentados en la sección 3.5 del capítulo 3.

Inicialmente se pensó diseñar un protocolo intermedio entre la aplicación de red y el controlador, pero durante el desarrollo se pudo comprobar de que el controlador *ODL* tenía una *Northbound API* que cumplía con nuestro propósito, por consiguiente, se decidió trabajar con dicha *API*. Y las políticas ingresadas por el usuario dentro de la plataforma son interpretadas por el controlador

con la ayuda de la *Northbound API* descrita en la sección 3.3 en el capítulo 3, y a su vez el controlador aplica la política dentro de la topología creada en el *Mininet* a partir de la *Southbound interface openflow* empleado en el capítulo 3 en la sección 3.4.

5.2 Recomendaciones

Como principal recomendación se recomienda investigar más a fondo las características de la versión del controlador *ODL,* ya que a lo largo del desarrollo nos encontramos con comportamientos indeseados que repercutieron con la implementación de la aplicación de forma correcta.

5.3 Trabajo a futuro

En primera medida en el presente trabajo se logró cumplir con el objetivo principal del proyecto, sin embargo, como trabajo a futuro se recomienda probar dicha plataforma en un entorno real, es decir empleando quipos del *PD* y *PC* físicos y analizar su comportamiento en un entorno de producción.

Así mismo, como trabajo a futuro adicional se pueden llevar a cabo investigaciones para diseñar una aplicación que permita instruir reglas de flujo que pueda abarcar más parámetros dentro del modelo de referencia *OSI,* con el fin de crear una aplicación mucho más completa y ser probada en topologías mucho más robustas y complejas que la utilizada en este proyecto.

Por último se deja el planteamiento como se mencionó en el capítulo 2, desarrollar un micro servicio dentro de *Apache Karaf* totalmente personalizado, que pueda ser instalado y ejecutado por el controlador *ODL,* que permita realizar alguna función de red dentro de la topología.

6. REFERENCIAS

A continuación, se presentan todas las citas bibliográficas utilizadas a lo largo de este documento.

Bibliografía

A Linux Foundation Collaborative Project. (2016). *openvswitch*. Recuperado el 8 de marzo de 2018, de openvswitch: http://www.openvswitch.org/features/

Arins, A. (2015). Firewall as a service in SDN OpenFlow network. *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)* (págs. 1 - 5). Riga, Latvia: IEEE.

Čejka, T. &. (2016). Configuration of open vSwitch using OF-CONFIG. *In Network Operations and Management Symposium* (págs. 883 - 888). Istanbul, Turkey: IEEE. doi:10.1109/NOMS.2016.7502920

Cisco Systems. (s.f.). *cisco*. Recuperado el 15 de Abril de 2018, de cisco: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html

Diego Kreutz, F. M. (2014). Software-Defined Networking: A Comprehensive Survey. (IEEE, Ed.) *Proceedings of the IEEE , 103*, 14-76.

Feamster, N., Rexford, J., & Ellen Zegura. (2 de Abril de 2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 87-98.

Google. (20 de Febrero de 2018). *YouTube*. Obtenido de YouTube: https://www.youtube.com/watch?v=SneKg7vAlzc

Lara, A. K. (1 de Octubre de 2014). Network Innovation using OpenFlow: A Survey. *IEEE communications surveys & tutorials*, 493-512. doi:10.1109/SURV.2013.081313.00105

Lara, A. K. (2014). Network innovation using openflow: A survey. *IEEE Communications Surveys & Tutorials*, 493-512. doi:10.1109/SURV.2013.081313.00105

Mininet Team. (2018). *mininet*. Recuperado el 7 de Marzo de 2018, de mininet: http://mininet.org/overview/

Nick Feamster, J. R. (Abril de 2014). The road to SDN: An intellectual history of programmable networks. *Computer Communication Review*, 87-98.

ONF. (2016). *ONF SDN Evolution version 1.0.* Open Networking Foundation.

Open Networking Foundation. (2015). *OpenFlow Switch Specification version 1.5.1.*

OpenDaylight Project. (2016 - 2018). *opendaylight*. Recuperado el 12 de Marzo de 2018, de opendaylight: http://docs.opendaylight.org/en/stable-carbon/getting-started-guide/introduction.html

OpenDaylight Project. (2016 - 2018). *OpenDaylight*. Recuperado el 12 de Marzo de 2018, de OpenDaylight: http://docs.opendaylight.org/en/stable-oxygen/getting-started-guide/introduction.html

OpenDaylight Project. (2016-2018). *opendaylight*. Recuperado el 13 de marzo de 2018, de opendaylight: http://docs.opendaylight.org/en/stable-oxygen/getting-started-guide/introduction.html

OpenDaylight Project a Series of LF Projects. (2018). *opendaylight*. Recuperado el 13 de Marzo de 2018, de opendaylight: https://www.opendaylight.org/what-we-do/odl-platform-overview

OpenDaylight. (s.f.). *wiki opendaylight*. Recuperado el 4 de Abril de 2018, de wiki opendaylight: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Restconf

OpenNetworking, F. (15 de Enero de 2018). *ONF*. Obtenido de https://www.opennetworking.org/sdn-definition/

OpenNetworkingFoundation. (2018). *opennetworking*. Recuperado el 5 de Marzo de 2018, de opennetworking: https://www.opennetworking.org/projects/mininet/

Oracle VM VirtualBox. (13 de Abril de 2018). *virtualbox*. Obtenido de virtualbox: https://www.virtualbox.org/manual/ch01.html

Sanchez-Velazquez, B. A. (2017, June). OpenFlow Communications and TLS Security in. *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom)* (pág. 560). IEEE. doi:10.1109

SDxCentral. (5 de Marzo de 2018). *SDxCentral*. Obtenido de sdxcentral: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/

Wajdy M. Othman, H. C.-M. (2017). Implementation and Performance Analysis of SDN Firewall on POX Controller. (págs. 1461 - 1466). Guangzhou, China: IEEE.

Wireshark Foundation. (2018). *wireshark*. Recuperado el 13 de Abril de 2018, de wireshark: https://www.wireshark.org/

7. APÉNDICE

En esta sección se presentan todos los apéndices

7.1 APENDICE A: Código de la aplicación JAVA

```
import javax.swing.JOptionPane;

/**
 * Trabajo de grado Ingeniería Telecomunicaciones
 *
 * Interfaz para aplicar políticas de seguridad en un entorno SDN
 *
 * @author Lucas Vásquez Agudelo
 *
 * Instituto Tecnológico Metropolitano
 *
 * 2018
 */
public class final1 {

    public void iniciar()
          {
        int opc=0,eleccion=0,conmutador=0,sw=1;

                              String iporigen=" ",ipdestino=" ",macorigen=" ",macdestino=" ";
```

```
JOptionPane.showMessageDialog(null,"Bienvenido a la plataforma para aplicar politicas de red SDN");

conmutador=Integer.parseInt(JOptionPane.showInputDialog("ingrese el numero switch al que quiere aplicar la politica: \n1. openflow:1\n2. openflow:2\n3. openflow:3"));

opc= Integer.parseInt(JOptionPane.showInputDialog("Ingrese el numero de la opcion con el tipo de politica: \n1. reenviar \n2. bloquear \n3. permitir"));

do
{

eleccion= Integer.parseInt(JOptionPane.showInputDialog("Ingrese el tipo de parametro por el que desea aplicar la politica: \n1. Ip-origen \n2. Ip-destino \n3. Mac-origen \n4. Mac-destino "));

switch(eleccion)

{
case 1:

iporigen= JOptionPane.showInputDialog("Ingrese la direccion IP origen:");

break;

case 2:
```

```
                                                    ipdestino=
JOptionPane.showInputDialog("Ingrese la direccion IP destino: ");
                                                    break;

                                                    case 3:


                                                    macorigen=
JOptionPane.showInputDialog("Ingrese la direccion MAC origen: ");
                                                    break;

                                                    case 4:


                                                    macdestino=
JOptionPane.showInputDialog("Ingrese la direccion MAC destino: ");
                                                    break;

                                                }


                                                 sw=
Integer.parseInt(JOptionPane.showInputDialog("Quiere ingresar otro parametro:  \n1.si \n2.no"));

                                                }
                                                while(sw==1);


                                JOptionPane.showMessageDialog(null,"La politica se aplicará al
conmutador: "+conmutador +" Su IP origen: "+iporigen +" Su IP destino: "+ipdestino +" Su MAC
origen: "+macorigen +" Su MAC destino: "+macdestino);
```

```
    }
}
```


7.3 Apéndice B: Solicitud *HTTP GET JSON*


```
{
   "nodes": {
      "node": [
         {
            "id": "openflow:2",
            "node-connector": [
               {
                  "id": "openflow:2:1",
                  "flow-node-inventory:port-number": "1",
                  "flow-node-inventory:name": "s2-eth1",
                  "flow-node-inventory:supported": "",
                  "flow-node-inventory:current-speed": 10000000,
                  "flow-node-inventory:current-feature": "copper ten-gb-fd",
                  "flow-node-inventory:configuration": "",
                  "flow-node-inventory:peer-features": "",
                  "flow-node-inventory:maximum-speed": 0,
                  "flow-node-inventory:advertised-features": "",
                  "flow-node-inventory:state": {
                     "link-down": false,
                     "blocked": false,
                     "live": false
                  },
                  "flow-node-inventory:hardware-address": "1E:EB:50:53:26:99",
                  "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
                     "transmit-errors": 0,
```

```
        "receive-drops": 0,
        "collision-count": 0,
        "duration": {
          "nanosecond": 915000000,
          "second": 2054
        },
        "transmit-drops": 0,
        "receive-frame-error": 0,
        "receive-over-run-error": 0,
        "receive-errors": 0,
        "packets": {
          "transmitted": 412,
          "received": 6
        },
        "receive-crc-error": 0,
        "bytes": {
          "transmitted": 35020,
          "received": 252
        }
      }
```


7.4 Apéndice C: *Petición HTTP PUT JSON*


```
        {
  "flow": [
    {
      "id": "1",
      "match": {
        "in-port": "1"
```

```
        },
        "instructions": {
          "instruction": [
            {
              "order": "0",
              "apply-actions": {
                "action": [
                  {
                    "order": "0",
                    "output-action": {
                      "output-node-connector": "2"
                    }
                  }
                ]
              }
            }
          ]
        },
        "flow-name": "h1-to-h2",
        "idle-timeout": "300",
        "table_id": "0"
      }
    ]
}
```

FIRMA ESTUDIANTE: Lucas Vásquez Agudelo.

FIRMA ASESOR 

FECHA ENTREGA: 03 de mayo de 2018

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____