UNIVERSIDAD NACIONAL DE LOJA

FACULTAD DE LA ENERGÍA, LAS INDUSTRIAS Y LOS RECURSOS NATURALES NO RENOVABLES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE REDES DEFINIDAS POR SOFTWARE (SDN), UTILIZANDO MININET COMO HERRAMIENTA DE SIMULACIÓN Y FÍSICAMENTE CON EL SWITCH SDN ZODIAC FX"

TRABAJO DE TITULACIÓN ESPECIAL PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTOR:

Geovanny de Jesús García Rojas

DIRECTOR:

Ing. Juan Gabriel Ochoa Aldeán Mg. Sc

Loja –Ecuador

2020

CERTIFICACIÓN

Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los Recursos Naturales No Renovables

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

Certificado Nro. 2020-001-FEIRNNR-UNL

Ing. Juan Gabriel Ochoa Aldeán Mg.Sc.
DIRECTOR DE TESIS.

CERTIFICA:

Que el estudiante: **Geovanny de Jesús García Rojas** con **C.I: 1105993701,** aprobó el Trabajo de Titulación en el **11vo Ciclo** correspondiente al período académico: **07-Octubre-2019** al **06-Marzo-2020**; respecto del desarrollo de su tesis de grado titulada **"DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE REDES DEFINIDAS POR SOFTWARE (SDN), UTILIZANDO MININET COMO HERRAMIENTA DE SIMULACIÓN Y FISICAMENTE CON EL SWITCH SDN ZODIAC FX",** opción de titulación escogida dentro del periodo académico de culminación de sus estudios, siendo las **15H00** del **13** de **febrero** del **2020**, se certifica que se ha cumplido con el cien por ciento (100%) del trabajo de titulación y está en condiciones de continuar con los procesos administrativos que correspondan.

Loja, 13 de febrero del 2020

Ing. Juan Gabriel Ochoa Aldeán Mg.Sc
DIRECTOR DE TESIS

Ciudad Universitaria "Guillermo Falconí Espinosa" Casilla letra "S"
Teléfono: 2545689 Ext. 133
dirección.cieyt@unl.edu.ec

AUTORÍA

Yo, **GEOVANNY DE JESÚS GACÍA ROJAS**, declaro ser autor del presente trabajo de tesis y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos o acciones legales, por el contenido de la misma.

Adicionalmente acepto y autorizo a la Universidad Nacional de Loja, la publicación de mi tesis en el Repositorio Institucional – Biblioteca Virtual.

Firma: 

Cédula: 1105993701

Fecha: 26/03/2020

CARTA DE AUTORIZACIÓN DE TESIS POR PARTE DEL AUTOR, PARA LA CONSULTA, REPRODUCCION PARCIAL O TOTAL Y PUBLICACION ELECTRONICA DEL TEXTO COMPLETO.

Yo, **Geovanny de Jesús García Rojas** declaro ser autor de la tesis **titulada "DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE REDES DEFINIDAS POR SOFTWARE (SDN), UTILIZANDO MININET COMO HERRAMIENTA DE SIMULACIÓN Y FÍSICAMENTE CON EL SWITCH SDN ZODIAC FX"**, como requisito para obtener el grado de **Ingeniero en Electrónica y Telecomunicaciones**; autorizo al Sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos, muestre al mundo la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Digital Institucional:

Los usuarios pueden consultar el contenido de este trabajo en RDI, en redes de información del país y del exterior, con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia de la tesis que realice un tercero.

Para constancia de esta autorización, en la ciudad de Loja a los treinta días del mes de abril del dos mil veinte.

Firma: 

Autor: Geovanny de Jesús García Rojas.

Cédula: 1105993701

Dirección: Loja (Barrio la Victoria, vía de Integración Barrial y vía a Payanchi)

Correo electrónico: gjgarciar@unl.edu.ec – geogarciarojas10@gmail.com

Celular: 0993393666

Director de Tesis: Ing. Juan Gabriel Ochoa Aldeán, M. Sc.

Tribunal de Grado: Ing. John Jossimar Tucker Yépez, Mg. Sc.

Ing. Renato Benjamín Torres Carrión Mg. Sc.

Ing. Marco Augusto Suing Ochoa Mg. Sc.

DEDICATORIA

Esta tesis la dedico primeramente a Dios por permitirme cada día tener una nueva oportunidad de vida para superarme como persona y como profesional, y a mis queridos padres José García y Mercedes Rojas por todo el sacrificio que han mostrado para permitirme cumplir todas mis metas, por el apoyo incondicional y la confianza que han depositado en mi durante el transcurso de mi carrera profesional.

A mis queridos hermanos por siempre apoyarme de una u otra forma para nunca desmayar en este proceso de formación profesional.

AGRADECIMIENTO

Quiero agradecer a Dios Todopoderoso por ser la guía y luz en mi vida durante el transcurso de mi formación profesional.

A mis amados padres por el constante apoyo económico y moral que me han brindado incondicionalmente durante toda mi vida como estudiante, por su confianza depositada en mí y por nunca dejarme rendir a pesar de las adversidades, por sus sabios concejos y por inculcarme valores que me ayudaron a crecer como persona de bien.

A mi Director de Tesis Ing. Juan Gabriel Ochoa Aldeán por el compromiso y apoyo mostrado hacia mi durante el desarrollo de mi tesis.

A todos los catedráticos que formaron parte de mi formación profesional por impartir sus conocimientos y darme la oportunidad de formarme como profesional.

A todos mis compañeros por brindarme su apoyo y sobre todo su amistad durante este proceso formativo.

A mi compañera de vida y amiga incondicional Gabriela Lima por su constancia y apoyo para nunca rendirme a pesar de los obstáculos que se presentan a diario.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 TÍTULO

"DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE REDES DEFINIDAS POR SOFTWARE (SDN), UTILIZANDO MININET COMO HERRAMIENTA DE SIMULACIÓN Y FÍSICAMENTE CON EL SWITCH SDN ZODIAC FX"

2 RESUMEN

El presente Proyecto de Titulación tiene como propósito diseñar e implementar un prototipo de redes definidas por software (SDN), utilizando Mininet como herramienta de simulación y físicamente con el Switch SDN Zodiac FX.

El desarrollo de la investigación está enfocado en desarrollar un escenario de prácticas orientadas a conocer el funcionamiento de SDN tanto de forma virtual como física, utilizando aplicaciones proporcionadas por el Controlador Ryu y OpenFlow como interfaz de comunicación SouthBound entre el Controlador y los elementos de la Red.

La primera sección corresponde al desarrollo de una serie de prácticas relacionadas con el simulador Mininet y el uso del Controlador Ryu, en dicha sección de llevan a cabo prácticas como: Instalación de herramientas de desarrollo y funcionamiento de las mismas; además, se ejecutan aplicaciones de Firewall, Router, IPv6 y Video. Todas estas prácticas se llevan a cabo en una PC donde se ejecuta el Controlador Ryu y el Simulador Mininet en el Sistema Operativo GNU/LINUX Ubuntu.

La segunda sección pertenece al desarrollo de prácticas utilizando el Switch Zodiac FX, donde de igual forma se llevan a cabo prácticas como: Introducción al uso de Zodiac FX y practicas similares a las desarrolladas en el simulador como es el caso de aplicaciones Firewall, Router, añadiendo Aplicaciones adicionales como Switch y Monitor de Tráfico.

ABSTRACT

The purpose of this Degree Project is to design and implement a prototype of software-defined networks (SDN), using Mininet as a simulation tool and physically with the Switch SDN Zodiac FX.

The development of the research is being focused on developing a scenario of oriented practices to know the operation of SDN both virtual and physical, using applications provided by the Ryu Controller and OpenFlow as a SouthBound communication interface among the Controller and the elements of the Network.

The first section corresponds to the development of a series of practices related to the Mininet simulator and the use of the Ryu Controller, in this section they perform practices such as: Installation of development tools and their operation; In addition, Firewall, Router, IPv6 and Video applications are run. All of these practices are carried out on a PC where the Ryu Controller and the Mininet Simulator are running in the Ubuntu GNU / LINUX Operative System.

The second section belongs to the development of practices using the Zodiac FX Switch, where practices such as: Introduction to the use of Zodiac FX and practices similar to those who were developed in the simulator are carried out, such as Firewall, Router applications, adding additional Applications such as Switch and traffic monitor.

3 INTRODUCCIÓN

Las Redes Definidas por Software (SDN: Software Defined Networks) han surgido en los últimos años como una propuesta para mejorar parámetros de velocidad, agilidad, disponibilidad, escalabilidad y bajos costos dentro de una red de datos, permitiendo la programación del comportamiento de una red a través de aplicaciones de software que utilizan API abiertas. SDN permite separar el plano de control del plano de datos; es decir, permite un control y administración centralizada de los dispositivos de red debido a que la inteligencia de la red está destinada al controlador y los dispositivos de red son únicamente encargados del manejo de flujos de datos.

Por lo mencionado anteriormente, se ha planteado el desarrollo e implementación de un prototipo de Redes Definidas por Software utilizando Mininet como herramienta de simulación y Zodiac FX como dispositivo físico.

El estudio empieza por adentrarse al concepto de SDN, sus principales características y beneficios, para de esta forma obtener los conocimientos necesario e implementar un escenario de prácticas que sirvan como base para futuros trabajos dentro de la Carrera de Ingeniería en Electrónica y Telecomunicaciones, respecto al tema antes mencionado. Además, se plantea como una solución a redes de datos tradicionales al incorporar Software en lugar de Hardware lo que trae consigo una multitud de beneficios, especialmente en el ámbito económico.

Se utiliza para el desarrollo de las prácticas el Laboratorio de Comunicaciones y Antenas de la Facultad de la Energía, las Industrias y los Recursos Naturales no Renovables de la Universidad Nacional de Loja, ya que ofrece las facilidades para el caso.

El presente proyecto de titulación tiene como objetivo general: ***Diseño e implementación de un prototipo de redes definidas por software (SDN), utilizando Mininet como herramienta de simulación y físicamente con el Switch SDN Zodiac FX***, de igual manera sus objetivos específicos son:

- ✓ Conocer a profundidad los conceptos, características y funcionamiento de las redes definidas por software (SDN).
- ✓ Entender el funcionamiento del software de virtualización Mininet para la simulación y programación de los elementos de Red.
- ✓ Desarrollar un compendio de prácticas de redes SDN basadas en un simulador como Mininet y físicamente a través del Switch SDN Zodiac FX, para futuros desarrollos en la carrera de Ingeniería en Electrónica y Telecomunicaciones.

4 REVISIÓN DE LITERATURA

4.1 Redes Definidas por Software (SDN)

4.1.1 Historia

Según Valencia, Santacruz, Becerra y Padilla (2015) la idea de programar redes se ha venido desarrollando desde hace algún tiempo atrás, existen avances y desarrollos anteriores que aportaron a lo que hoy son las SDN:

- ✓ **SOFTNET:** surgió alrededor de los años 80, una red multisalto cuya innovación fue que en cada paquete de datos se incluía comandos que los nodos iban ejecutando conforme los iban recibiendo.
- ✓ **DCAN (Devolved Control of ATM Networks):** tuvo lugar a mediados de la década de 1990, su objetivo era desarrollar una infraestructura idónea para el control y gestión escalable de redes de cajeros automáticos. DCAN asume un protocolo simplificado entre el gestor y la red, que es lo que sucede hoy en día con OpenFlow.
- ✓ **NETCONF:** desarrollado en 2006 por el Grupo de Trabajo de Ingeniería de Internet IETF, es un protocolo de gestión para realizar modificaciones en dispositivos de red. Mediante este protocolo, los dispositivos de red presentaban una API que permitía que los datos de configuración puedan ser enviados y recuperados.
- ✓ **ETHANE:** en 2006 definió una arquitectura nueva para redes empresariales. Su enfoque consistió en el uso de un controlador centralizado para gestionar la política y la seguridad en una red.

En el año 2007 un grupo de trabajo de la Universidad de Standford formado por Nick McKeown, Scott Shenker y Martín Casado desarrollaron OpenFlow y fundaron Nicira, una compañía de virtualización de redes. En 2011 Scott Shenker y Nick McKeown fundaron la Open Networking Foundation (ONF), que era una organización que buscaba impulsar el uso de OpenFlow, la creación de estándares y el surgimiento de SDN más allá de las Universidades. En julio de 2012 VmWare, una de las compañías líderes en virtualización de

servidores compro Nicira incorporando la virtualización de redes en su gama de productos (Roncero, 2016).

4.1.2 Generalidades

El constante crecimiento de datos que circulan por las redes y las características con las que cuentan en la actualidad los protocolos y dispositivos de red, dan lugar a que las redes IP tradicionales resulten complejas de manejar. Una de las causas que origina esta problemática obedece al hecho de que los dispositivos de red son cajas negras; es decir, contienen un firmware con las configuraciones de enrutamiento definidas previamente por el fabricante, lo que dificulta a los administradores y operadores de red realizar gestiones como distribución de carga, privilegios de transmisión, cambios de métricas en los protocolos, rutas alternas, y otros (Valencia et al., 2015).

Para alcanzar políticas de red de alto nivel, los operadores de red deben configurar cada dispositivo de forma individual utilizando comandos específicos dados por el proveedor de equipos. Además de la complejidad de la configuración, los entornos de red deben soportar la dinámica de fallas y adaptarse a los cambios de carga. Los mecanismos automáticos de reconfiguración y respuesta son prácticamente inexistentes en las redes IP actuales. Hacer cumplir las políticas requeridas en un entorno tan dinámico es, por lo tanto, un gran desafío (Kreutz et al., 2014).

En SDN, los dispositivos de red se convierten en simples dispositivos de reenvío de paquetes, mientras que la lógica de control se implementa en el controlador. Este cambio de paradigma trae varios beneficios en comparación con los métodos heredados. Primero, es mucho más fácil introducir nuevas ideas en la red a través de un programa de software. En segundo lugar, SDN presenta los beneficios de un enfoque centralizado para la configuración de la red; es decir, los operadores no tienen que configurar todos los dispositivos de red individualmente para realizar cambios en el comportamiento de la red (kim & Feamster, 2013).

4.1.3 Definición

Las Redes Definidas por Software o SDN son una arquitectura dinámica, administrable, confiable y escalable, la cual otorga las funciones de manejo del flujo de paquetes a una aplicación de software llamada controlador, función que en redes tradicionales son llevadas a cabo por dispositivos físicos de red (Bonilla, 2016).

SDN es un ejemplo o modelo en la forma de gestionar una red. Separa la gestión de la conmutación, siendo su objetivo principal desacoplar la infraestructura de la red de las aplicaciones y servicios que ésta proporciona (Roncero, 2016).

Las Redes Definidas por Software comprenden una tecnología en la cual el plano de datos se separa del plano de control y utilizan controladores basados en software, que cumplen la función de gestionar el flujo de datos entre el controlador y los conmutadores (España, 2016).

SDN permite gestionar el plano de control utilizando software y el protocolo OpenFlow, siendo posible la administración de este plano mediante el uso de equipos servidores llamados controladores, mientras que los equipos de conectividad siguen manejando el plano de datos (Morillo, 2014).

4.2 Arquitectura SDN

El factor más relevante en las Redes Definidas por Software es la separación del plano de datos y el plano de control, debido a esta división SDN establece una arquitectura compuesta por tres capas: Capa de Aplicación, Capa de Control y Capa de Infraestructura, como se muestra en la figura 1 (Bonilla, 2016).

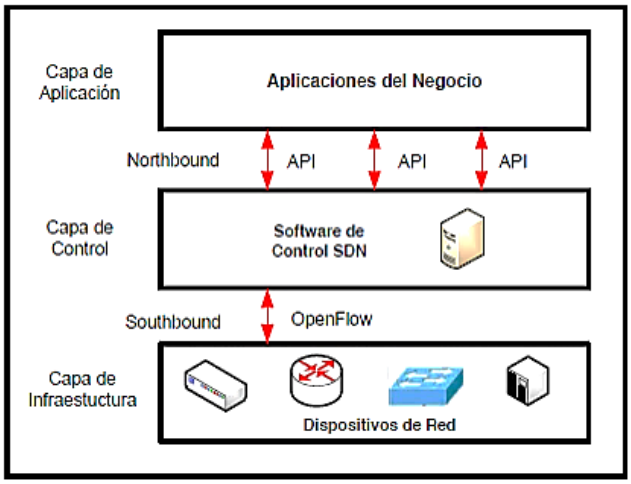


Figura 1. Arquitectura SDN

Fuente: (Bonilla, 2016)

En SDN la inteligencia de la red esta centralizada en los controladores de manera lógica, los mismos que mantienen una visión general de la red. Como resultado, la red aparece ante las aplicaciones de control como un solo Switch lógico, lo cual reduce la complejidad de operación y permite mejorar la escalabilidad del controlador (Morillo, 2014).

4.2.1 Capa de Infraestructura

La capa de infraestructura la componen los dispositivos de red como Switch y Routers, que en SDN se pueden configurar a través de controladores por medio de reglas sofisticadas de supervisión de tráfico para el cambio y políticas de seguridad. Esta capa se encuentra relacionada con el Plano de Datos (Valencia et al., 2015).

Los dispositivos que conforman la capa de infraestructura administran las tablas de flujo en función de las indicaciones dadas por el controlador (Bonilla, 2016).

4.2.2 Capa de Control

La capa de control está compuesta por uno o más controladores SDN que se encargan de coordinar políticas de administración, además de la comunicación con las otras capas de la arquitectura SDN, esto lo hacen utilizando interfaces y protocolos específicos (Gonzales, Flauzac & Nolot, 2018).

Es la capa intermedia, que tiene una visión global de la red y además incorpora el Sistema Operativo de Red (NOS). Se encarga de tomar decisiones y programar tablas de flujo de elementos de la capa inferior para controlar y gestionar el desvío de tráfico, el enrutamiento de paquetes, la prioridad de paquetes y otros (Valencia et al., 2015).

La Capa de Control establece el comportamiento de los flujos de red; el controlador permite la transmisión de instrucciones hacia los dispositivos de red a través de las interfaces de SouthBound, la más conocida OpenFlow. Esta capa se encuentra relacionada con el plano de Control (Bonilla, 2016).

4.2.3 Capa de Aplicación

Es la capa superior de la Arquitectura SDN, está compuesta por las aplicaciones creadas por los usuarios, que son las que permiten personalizar las redes. Incorporan las NorthBound APIs, y es aquí donde se produce un cambio significativo respecto a las redes tradicionales (Valencia et al., 2015).

Esta capa es conocida también como capa de negocio, tiene relación directa con los servicios o aplicaciones de alto nivel que requieren los usuarios, entra las cuales destacan características como QoS (Calidad de Servicio), Firewall, y otros (Bonilla, 2016).

Las capas de la arquitectura SDN, según España (2016), interactúan mediante las interfaces abiertas: SouthBound “Hacia el sur” y NorthBound “Hacia el norte”.

SouthBound: permite comunicar el controlador SDN y los dispositivos de red en la capa inferior. El elemento principal de esta interfaz es el protocolo OpenFlow, que permite implementar las soluciones SDN.

NorthBound: permite la comunicación del controlador con las aplicaciones. Permite, incluir funciones básicas de red, como ruteo, manejo de tráfico de datos y seguridad para administración de servicios de red en la nube.

4.3 OpenFlow

OpenFlow es una tecnología que deriva de una investigación sobres redes virtuales propuesta por la Universidad de Standford, basado en Ethane, que era un producto que contenía algunas funciones que OpenFlow utiliza en la actualidad. OpenFlow acerca conceptos ya conocidos como separación de planos y, además se encarga de agregar otros tantos como el uso de estándares de comunicación entre controladores y dispositivos de red y proporciona una API a los administradores de red para programar la red SDN (Valencia, 2015).

OpenFlow define un protocolo estándar que determina las acciones de reenvió de paquetes en elementos de red como Switch, Router y puntos de acceso inalámbricos. Las acciones y reglas instaladas en el hardware de red son asumidas por un elemento externo, denominado controlador, y puede implementarse en un servidor común (Rothenberg, Nascimento, Salvador & Magalhães, 2010), como lo indica la Figura 2.

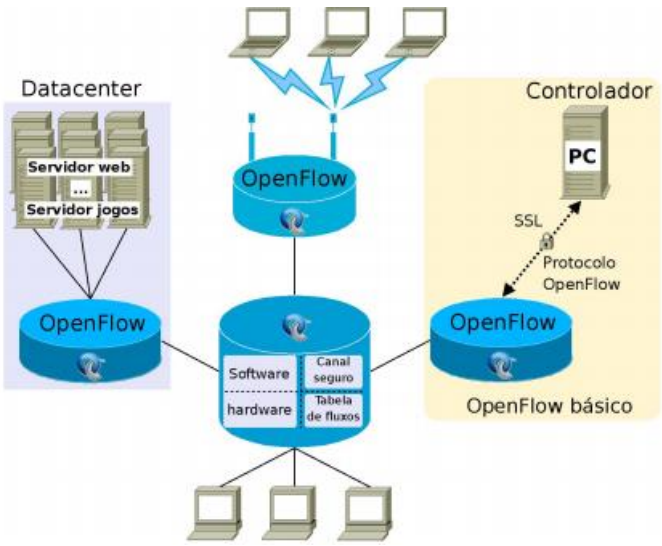


Figura 2. Ejemplo de una red con OpenFlow

Fuente: (Rothenberg et al., 2010)

El protocolo OpenFlow se implementa en dispositivos que forman parte de la infraestructura de red y el controlador SND. Para identificar el tráfico de red, usa el concepto de flujos y se basa en reglas predefinidas, que se pueden programar de forma estática o dinámica en el controlador, lo que permite a la red responder en tiempo real a nivel de aplicación, usuario y sesión (España, 2016).

En la tabla 1 se resume varias versiones de OpenFlow y sus características más relevantes:

Tabla 1. Versiones de OpenFlow y sus Características.

	OPENFLOW 1.0	OPENFLOW 1.1	OPENFLOW 1.2	OPENFLOW 1.3 Y 1.4
PUERTO DE ENTRADA	**X**	**X**	**X**	**X**
METADATOS		**X**	**X**	**X**
ETHERNET: SRC,DST,TYPE	**X**	**X**	**X**	**X**
IPV4: SRC,PROTO,TOS	**X**	**X**	**X**	**X**
TCP/UDP: SRC PORT, DST PORT	**X**	**X**	**X**	**X**
MPLS: ETIQUETA, TIPO DE TRAFICO		**X**	**X**	**X**
OPENFLOW EXTENSIBLE MATCH (OXM)			**X**	**X**
IPV6: SRC, DST, FLOW LABEL, ICMPV6			**X**	**X**
IPV6 EXTENSION HEADERS				**X**

Fuente: (Valencia, 2015)

Las SDN basadas en OpenFlow se pueden implementar conjuntamente con las redes existentes, tanto de forma física como virtual. Los elementos de red pueden soportar el reenvío de tráfico basado en OpenFlow, así como también el reenvió tradicional, lo que facilita al usuario introducir tecnologías basadas en OpenFlow de manera progresiva, incluso si se tiene un escenario de red de múltiples proveedores (Morillo, 2014).

4.3.1 Switch OpenFlow

Un Switch OpenFlow se compone, según Bonilla (2016) de un canal de comunicación seguro para interactuar con el controlador, de una o más tablas de flujo y de una tabla de grupo, como se muestra en la figura 3.

- ✓ **Tabla de Flujo:** a través del protocolo OpenFlow, el controlador puede agregar, actualizar y eliminar en las tablas entradas de flujo, cada tabla consta de dos campos para emparejamiento, un campo corresponde a los contadores de actividad y el otro campo corresponde a un conjunto de instrucciones que se aplican a los paquetes coincidentes.
- ✓ **Tabla de Grupo:** en esta tabla se establece los conjuntos de elementos que pueden llevar a cabo la misma acción. Son de gran ayuda al momento de implementar reglas para tráfico multicast, aunque también son usadas para tráfico unicast con similares características.
- ✓ **Canal Seguro:** corresponde a la interfaz encargada de conectar el Switch OpenFlow al controlador, el servidor controlador administra, recibe eventos y envía paquetes al Switch a través del canal seguro.

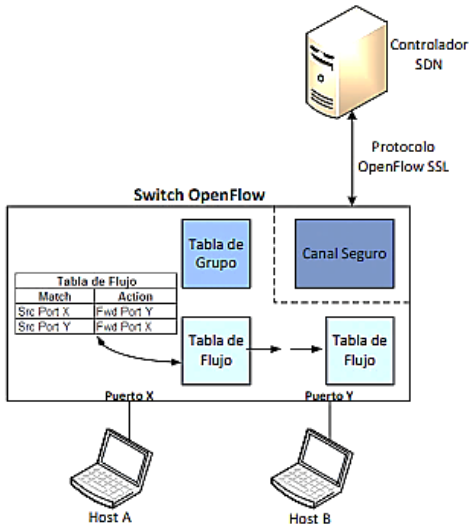


Figura 3. Elementos Switch OpenFlow

Fuente: (Bonilla, 2016)

Roncero (2016) recalca que, un Switch OpenFlow cuenta con varias tablas de flujo, en las cuales se realiza la búsqueda de coincidencias con los paquetes entrantes, y cada entrada en una tabla de flujos se compone de tres campos:

- Cabecera, que define el flujo de paquetes.
- Contadores, que contabilizan los paquetes coincidentes, se actualizan por flujo, por tabla, por puerto y por cola.
- Acciones a realizar cuando existe coincidencia entre un paquete y una tabla de flujo.

La figura 4 muestra el procesado de un paquete por parte de OpenFlow Switch.

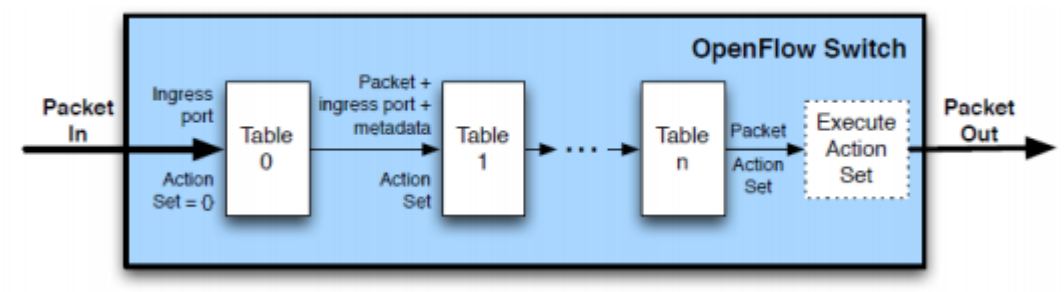


Figura 4. Procesado de un Paquete en un Switch OpenFlow

Fuente: (Roncero, 2016)

El procesado de un paquete empieza, cuando este entra al Switch OpenFlow, se examina desde la primera tabla y en caso de existir coincidencias, se realiza la acción configurada previamente en el Action Set (conjunto de acciones a ejecutar definidas en el Switch), y se evalúa el paquete en la tabla siguiente. Una vez evaluadas todas las tablas y en caso de no existir coincidencias, el paquete será devuelto al controlador, que indicará al Switch que acción tomar (Roncero, 2016).

4.3.2 Clasificación de los Switch OpenFlow

Es útil, según Morillo (2014) clasificar a los Switch OpenFlow en tres grupos: Switch dedicados, Switch habilitados para OpenFlow y Switch Tipo “0”.

4.4 Switch OpenFlow Dedicados

Contiene las rutas de datos para el reenvío de paquetes entre puertos, de la misma forma como se define en el servidor controlador. La tabla de flujo está controlada a través de un canal seguro y los flujos se definen ampliamente y se limitan únicamente por las capacidades de aplicación en el controlador (Morillo, 2014).

En la figura 5 se muestra un ejemplo del Switch OpenFlow dedicado.

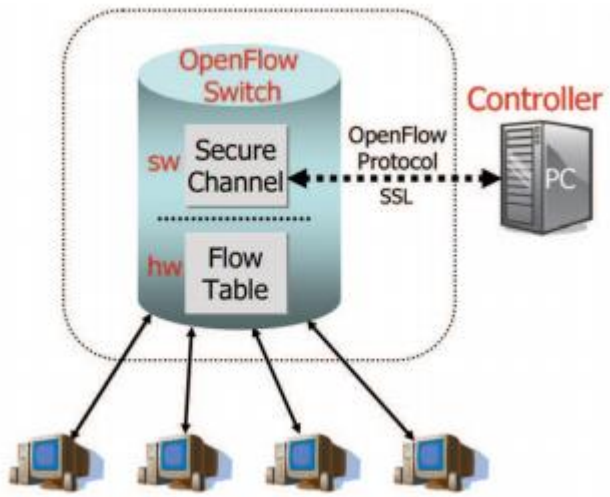


Figura 5. Switch OpenFlow Dedicado

Fuente: (Pereira & Gamess, 2017)

Cada entrada de flujo tiene asociado consigo una acción simple, y según Morillo (2014) las tres acciones básicas que todo Switch OpenFlow dedicado debe soportar son:

- **Reenviar,** el flujo de paquetes a un puerto determinado, que permitirá enviar los paquetes a través de la red.
- **Encapsular** y reenviar estos paquetes de flujo a un controlador. El paquete se entrega ala canal seguro, donde se realiza la encapsulación y se envía al controlador, el cual decide si se debe añadir la entrada de flujo a la tabla para su procesamiento.
- **Descartar** el flujo de paquetes. Se puede hacer este proceso para garantizar la seguridad en la red, frenar los ataques de denegación de servicio o reducir el tráfico de difusión.

4.4.1.1 *Switch Habilitados para OpenFlow*

Algunos conmutadores, Routers y puntos de acceso tendrán una mejora con la integración de OpenFlow, añadiendo la tabla de flujo especificada por OpenFlow, el canal seguro y el protocolo OpenFlow en sí. Generalmente, la tabla de flujo utilizara el hardware existente, mientras que el canal seguro y el Protocolo OpenFlow se adaptaran de tal forma que puedan ser ejecutados en el Sistema Operativo del Switch (Morillo, 2014).

La figura 6 muestra la red con Switch habilitado para OpenFlow y puntos de acceso.

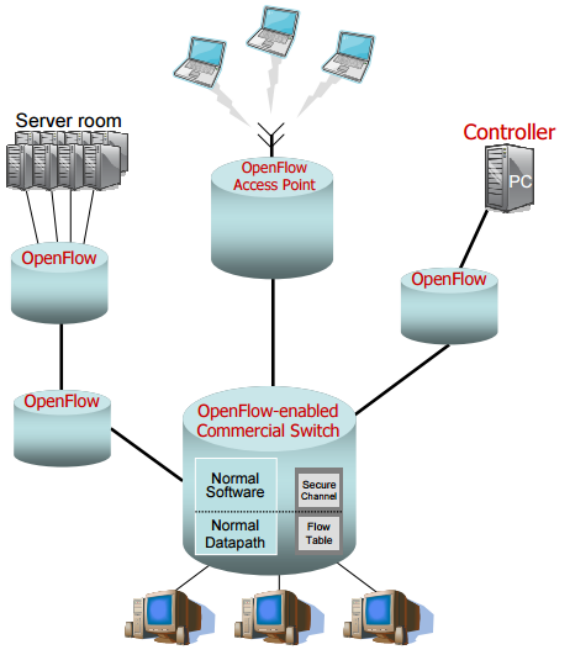


Figura 6. Red con Switch habilitado para OpenFlow

Fuente: (Dignan, 2012)

Todas las tablas de flujo funcionan usando el mismo controlador. El Switch puede ser controlado por dos o más controladores a través del protocolo OpenFlow, que permitirá incrementar el rendimiento del sistema (Morillo, 2014).

4.4.1.2 *Switch Tipo "0"*

Presentan los requerimientos mínimos que les permitan ajustarse a las funcionalidades de un Switch OpenFlow. Si un Switch soporta los formatos de encabezado y las acciones básicas mencionadas anteriormente, se lo llama Switch "Tipo 0" (Morillo, 2014).

4.4.2 Servidor Controlador

Un controlador funciona como el Sistema Operativo de la red y se encarga de controlar todas las comunicaciones entre aplicaciones y elementos de red. Se encarga de traducir los requerimientos de la capa de aplicación a los elementos de red, y además proporciona información relevante a las aplicaciones SDN (España, 2016).

Este Software se sitúa en la capa de control, en el cual realiza la función de centralizar la red, para ello utiliza el protocolo OpenFlow como lenguaje de comunicación con los distintos nodos de la red. El controlador se encarga de introducir, gestionar y controlar los nodos de red a través de reglas predefinidas (Álvarez, 2017).

El controlador actúa sobre el plano de control, como interfaz entre las aplicaciones de control de red y la red como tal. OpenFlow cuenta con una gran flexibilidad que permite generar varios escenarios para su diseño (Tinajero, 2016).

En la figura 7 se muestra dos formas de conexión del controlador.

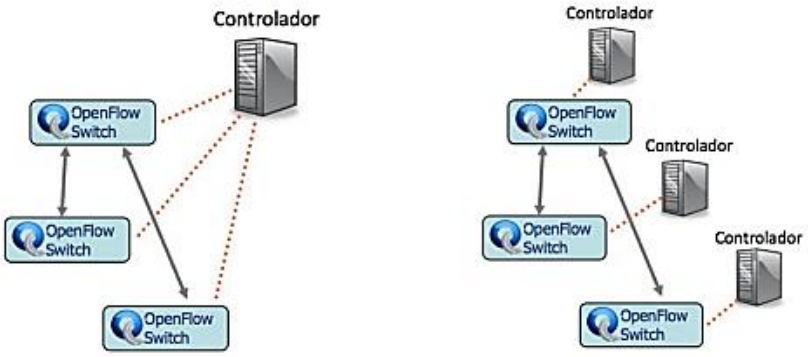


Figura 7. Formas de Conexión de Controladores OpenFlow.

Fuente: (Tinajero, 2016)

Según Tinajero (2016), existen dos formas de conexión de un controlador: el centralizado en donde existe un único controlador y todos los conmutadores se conectan a él, y el controlador distribuido donde existen dos o más controladores en los que la gestión de red se divide para los nodos de la red.

4.4.3 Tipos de controladores

La tabla 2 resume algunos de los controladores de código abierto y las características más relevantes de los mismos.

Tabla 2. Tipos de Controladores y sus Características.

	NOX	POX	RYU	OpenDaylight (ODL)	ONOS	Floodlight
Soporte OpenFlow	v1.0	v1.0	v1.0, v1.2, v1.3, v1.4 v1.5	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3	v1.0, v1.3
Lenguaje	C++	Python	Python	Java	Java	Java + Cualquier lenguaje que soporte REST
Plataforma	Linux	Linux, Mac Os, Windows	Linux	Linux	Linux	Linux, Mac Os, Windows
Soporte OpenStack	No	No	Si	Si	Si	Si
Multiproceso	Si	No	No	Si	Si	Si
Código Abierto	Si	Si	Si	Si	Si	Si
API REST	No	No	Si	Si	Si	Si
Virtualización	Mininet y Open vSWITCH	Mininet y Open vSWITCH	Mininet y Open Vswitch	Mininet y Open vSWITCH	Mininet y Open vSWITCH	Mininet y Open vSWITCH

Fuente: (Bonilla, 2016)

4.4.4 NOX (Network Operating System)

Es uno de los primeros controladores con el cual se realizó pruebas en redes SDN Y que obtuvo mayor rendimiento. Las primeras pruebas que se hizo con NOX arrojaron respuestas a 30Krps (Miles de solicitudes por segundo); sin embargo, las mediciones que se hicieron de un tráfico promedio en una red con 100 Switches demostraron que este controlador no era capaz de administrar una red de grandes dimensiones (Gómez, 2013).

La figura 8 muestra los Componentes de una red basada en NOX.

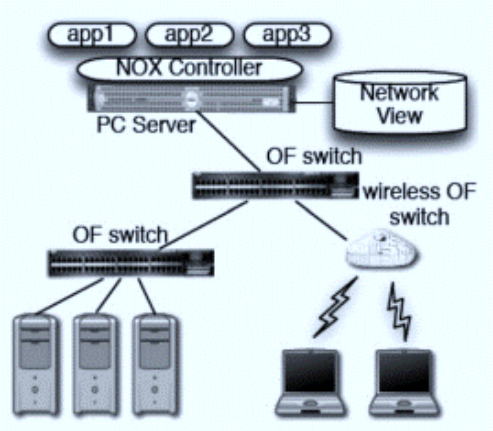


Figura 8. Componentes de una red basada en NOX.

Fuente: (Gómez, 2013)

Ante las limitaciones de NOX, surgió NOX-MT, un controlador que trabaja con multi-hilos, donde es directamente proporcional la relación que existe entre el número de hilos y las solicitudes atendidas. NOX-MT utiliza diversas técnicas de optimización; entre ellas la Entrada/Salida de procesos ejecutados por lotes, lo que brinda mayor rendimiento, según las pruebas efectuadas, esta evolución desencadeno un rendimiento 6 veces superior al NOX; esto en un CPU de un solo núcleo. En una CPU de 8 núcleos se alcanzó un rendimiento 33 veces superior (Gómez, 2013).

4.4.5 POX

POX es un controlador creado para cubrir las necesidades de redes SDN utilizando lenguaje de programación de Python, se lo usa para Sistemas Operativos basados en Windows, Mac OS y Linux (Contreras, 2014).

Las características más relevantes del controlador POX, según Contreras (2014) son las que se mencionan a continuación:

- Lenguaje de OpenFlow basado en Python
- Detección de topologías de red y selección de rutas de acceso
- Interfaz gráfica para el usuario y componentes de visualización
- Desarrollado para el manejo del protocolo OpenFlow en versión 1.0

POX es la primera plataforma creada para construir aplicaciones para el control de la red en OpenFlow. Fue creada para el rápido desarrollo y para controlar el software de la red, a nivel básico y con un rendimiento bastante similar a NOX (Serrano, 2016).

4.4.6 OpenDaylight

OpenDaylight (ODL) es una plataforma abierta para programar SDN y Virtualización de funciones de Red (Network Functions Virtualization NFV) en redes de cualquier tamaño o escala. Este controlador soporta integración OpenStack y está escrito en lenguaje de programación Java. OpenDaylight combina componentes que incluyen controlador, interfaces y aplicaciones. No está limitado a OpenFlow, sino que está abierta al uso de otros protocolos como BGP-PCEP y NETCONF (Sotelo, 2015).

Sotelo (2015), manifiesta que la arquitectura base de OpenDaylight se compone de tres capas:

- Interfaces para los protocolos de tipo SouthBound (SB)
- Capa de adaptación de servicios (Service Adaption Layer o SAL)
- Interfaces de tipo NorthBound (NB) para comunicarse con las aplicaciones

Esta arquitectura ha presentado cambios y mejores en las diferentes versiones de OpenDaylight, especialmente en *Helium* que es la versión más reciente.

Esto se aprecia en la siguiente figura.

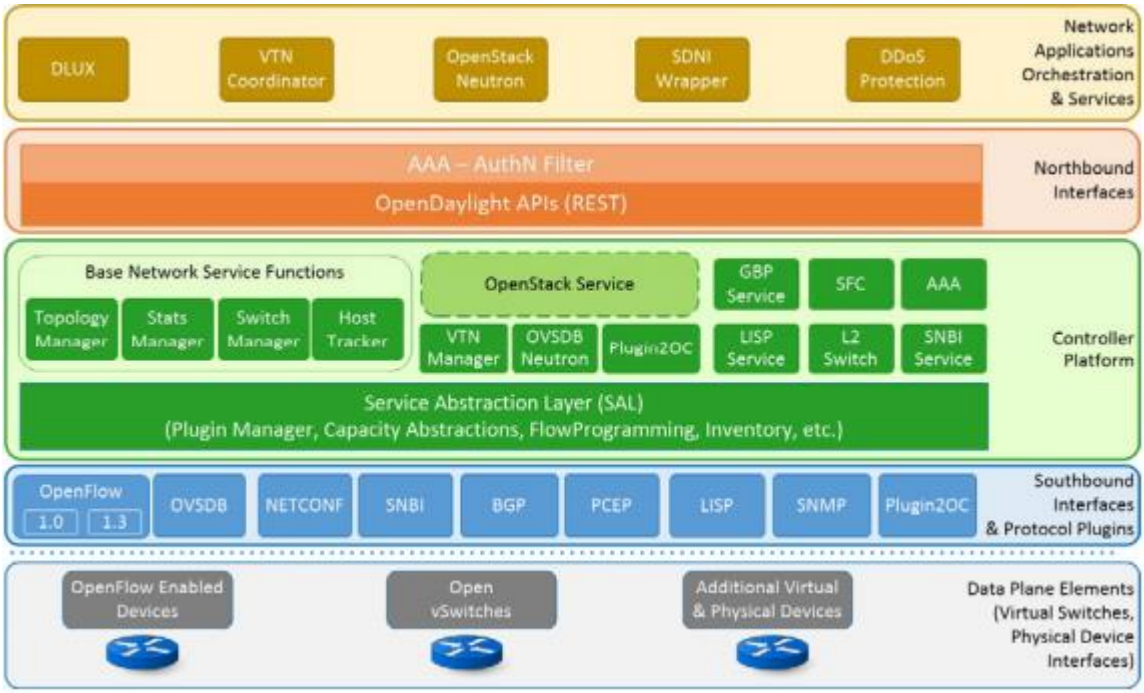


Figura 9. Arquitectura OpenDaylight

(Sotelo, 2015)

4.4.7 Floodlight

Es un controlador SDN de código abierto que cuenta con una licencia Apache, está escrito en lenguaje de programación Java y soporta el protocolo OpenFlow como interfaz de Comunicación SouthBound con los elementos de la red. Su arquitectura es modular y proporciona un conjunto de aplicaciones en Java integrados en el controlador (Sotelo, 2015).

La figura 10 muestra la arquitectura Floodlight y la relación entre el controlador, las aplicaciones que presentan interfaces de programación en Java y las aplicaciones accesibles dese interfaces REST.

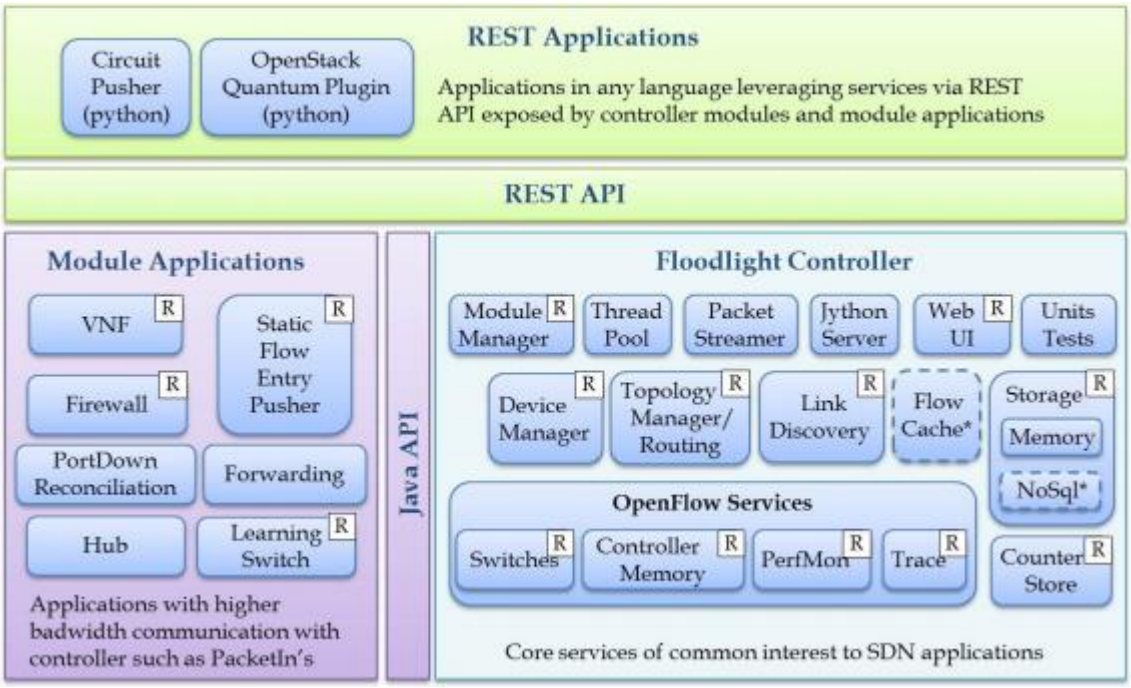


Figura 10. Arquitectura Foodlight

(Sotelo, 2015)

4.4.8 Ryu

Ryu es un software de código abierto que se basa en componentes definidos por un framework de redes, está basado en lenguaje de programación Python y proporciona al igual que otros controladores componentes de software utilizando APIs bien definidas, disponibles para que los desarrolladores creen aplicaciones de control y gestión de red. Una de las ventajas más importantes de Ryu es que puede soportar múltiples protocolos SouthBound para gestión de dispositivos, tales como OpenFlow, NETCONF, OF-Config, y otros (Tinajero, 2016).

Es un controlador diseñado como una aplicación monoproceso que trae consigo un problema de rendimiento, pues al no contar con una arquitectura multiproceso no permite generar aplicaciones paralelas de forma natural, lo que conlleva a que su ejecución sea lineal. Por el contrario, la ventaja que trae consigo esta arquitectura es que simplifica la implementación de aplicaciones y prototipos que no requieren contar con un rendimiento alto (Álvarez, 2017).

Figura 11. Manejador de Procesos Ryu.

Fuente: (Álvarez, 2017)

Ryu implementa diversas funcionalidades, entre ellas manejadores de procesos (ver figura 9), visores de topología e incluso generadores de paquetes. Cada aplicación en Ryu recibe eventos utilizando una cola FIFO, preservando de esta forma la llegada de los diferentes mensajes, a esta forma de gestión de paquetes Ryu define como Manejadores de Eventos (Álvarez, 2017).

Yánez & Gallegos (2015) mencionan algunas de las características del Controlador Ryu:

- Cuenta con interfaz OpenFlow propia, que facilita la creación de nuevas aplicaciones de control.
- Funciona bajo sistema operativo Linux de código abierto y está disponible para Licencia Apache2.0.
- Utiliza componentes ya existentes y reusables; además, que permite modificar e implementar nuevos componentes.
- La creación de la red virtual es completamente independiente del controlador y la ejecución del mismo.
- Utiliza librerías específicas, dependiendo el desarrollo de aplicación que requieran conexiones, lectura de datos, llamadas, y otros.

4.4.8.1 *Arquitectura Ryu*

El controlador Ryu puede crear y enviar mensajes OpenFlow, escuchar mensajes asíncronos como Flow-removed y Packet-in, así como también realizar un análisis de paquetes entrantes (Tinajero, 2016). La figura 12 muestra los elementos que componen la arquitectura del controlador Ryu.

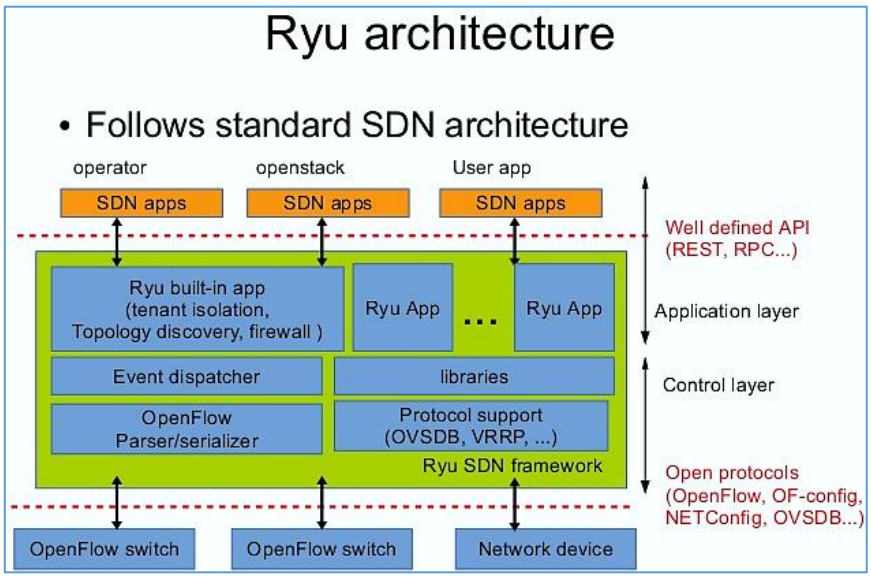


Figura 12. Arquitectura RYU.

Fuente: (Yamahata, 2013)

4.4.8.2 *Componentes Ryu*

Ryu proporciona instrucciones y comandos para poder ejecutar aplicaciones y comandos llevados a cabo en el controlador. Consta de componentes útiles para aplicaciones SDN, los componentes existentes pueden ser modificados e implementar nuevos (Guerrero, 2017). La tabla 3 muestra los componentes incluidos en Ryu.

Tabla 3. Componentes del Controlador Ryu.

COMPONENTES RYU

Ejecutables	bin / ryu-manager	El ejecutable principal.
Componentes de Base	El ejecutable principal.	-Cargar aplicaciones Ryu. -Proporcionar contextos a las aplicaciones Ryu. -Ruta de mensajes entre aplicaciones de Ryu.
Controlador OpenFlow	ryu.controller.controller	Maneja las conexiones de los interruptores. Genera y enruta eventos a entidades apropiadas como aplicaciones Ryu.
	ryu.controller.dpset	Planeado para ser reemplazado por ryu/topología.
	ryu.controller.ofp_event	Definiciones de eventos de OpenFlow.
	ryu.controller.ofp_handler	Manejo básico de OpenFlow, incluida la negociación.
Aplicaciones Ryu	ryu.app.simple_switch	Una implementación de switch de aprendizaje OpenFlow 1.0 L2.
	ryu.topology	Cambiar y vincular el módulo de descubrimiento. Planeado para reemplazar ryu/controller/dpset.
Bibliotecas	ryu.lib.packet	Implementaciones de descodificadores/codificadores de protocolos populares como TCP/IP.
	ryu.lib.ovs	Ovsdb biblioteca de interacción.
	ryu.lib.of_config	OF-Config implementación
	ryu.lib.netconf	Definiciones de NETCONF utilizadas por ryu/lib/of_config.

Fuente: (Guerrero, 2017)

El usuario además de disponer los componentes mencionados en la tabla anterior; puede desarrollar sus propios componentes para el escenario que requiera. El directorio donde se alojan los componentes nuevos es /ryu/app para su posterior uso y ejecución (Yánez, 2015).

4.4.8.3 *Aplicaciones Ryu*

Ryu se distribuye con una variedad de aplicaciones como: simple_switch, router, isolation, firewall, GRE Tunnel, VLAN, entre otros. Estas aplicaciones son entidades de un único subproceso, las cuales implementan una gran variedad de funcionalidades, mismas que envían mensajes asincrónicos entre sí denominados eventos (Tinajero, 2016).

El funcionamiento de esta arquitectura se muestra en la figura 13.

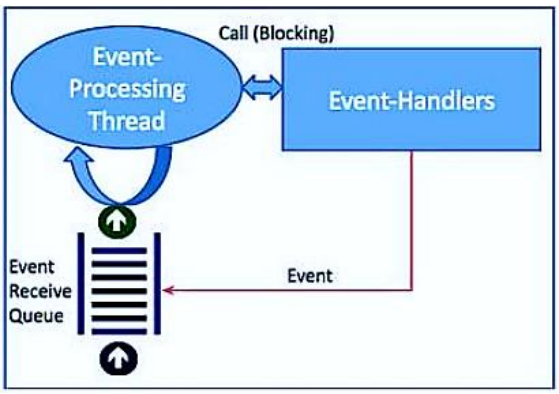


Figura 13. Funcionamiento de la Arquitectura de Ryu.

Fuente: (Tinajero, 2016)

Cada aplicación Ryu consta de una cola de recepción de eventos, la cual se conoce como FIFO (primero en entrar, primero en salir) y conserva el orden de los eventos. Cada aplicación Ryu consta de un hilo para procesar un evento, el subproceso sigue drenando la cola de recepción al momento de eliminar un evento y llamar al controlador apropiado para el tipo de evento que se lleve a cabo; se debe tener cuidado al momento de llamar al controlador de eventos, ya que puede existir un bloqueo. Si esto sucede, no se procesarán más eventos para la aplicación Ryu (Guerrero, 2017).

4.4.8.4 *Estructura de Código en Ryu*

En la carpeta */ryu* se encuentra alojado el código principal del controlador. Tinajero (2016) menciona las funcionalidades de los componentes claves de Ryu:

- **app/:** posee un conjunto de aplicaciones que se ejecutan sobre el controlador.
- **base/:** posee la clase base para aplicaciones Ryu.
- **controller/:** conformada por un conjunto de archivos necesarios para el manejo de funciones OpenFlow.
- **lib/:** posee un conjunto de librerías de paquetes para el análisis de diferentes cabeceras del protocolo y una librería de OF-CONFIG. Además, cuenta con analizadores para NetFlow y Sflow.
- **ofproto/:** posee información detallada del protocolo OpenFlow y de analizadores relacionados para soportar varias versiones del protocolo antes mencionado (v1.0, v1.2, v1.3, v1.4).
- **topology/:** contiene código para realizar el descubrimiento de topología relacionada con Switch OpenFlow y maneja información asociado con puertos, enlaces, y otros. Internamente utiliza el protocolo LLDP (Link Layer Discovery Protocol).

4.5 Mininet

Existe una gran variedad de herramientas de simulación como: Kiva, Packet Tracer, NS2, OMNet++, NS-3, ESTINET en su mayoría compatibles con el protocolo OpenFlow, dentro de estas se encuentra Mininet. Es una plataforma escalable para simulación de SDN mediante el uso de virtualización permitiendo la creación e interacción de distintas topologías de red. Permite ejecutar un conjunto de hosts, Switch, Routers, links y controladores sobre un kernel Linux (España, 2016).

Mininet es un emulador que surge como una solución para el despliegue de redes sobre los limitados recursos de un ordenador simple o máquina virtual. En una red OpenFlow emulado con el uso de Mininet, una aplicación de controlador real OpenFlow puede ejecutarse

en una maquina externa o en el mismo equipo en el que se emulan los hosts virtuales (Valencia et al., 2015).

En la actualidad los simuladores que permiten el prototipado y la experimentación con SDN más conocidos son NS-3, EstiNet y Mininet (Valencia et al., 2015). La tabla 4 muestra una comparativa entre los tres emuladores, destacando sus características más relevantes.

Tabla 4. Comparación entre Estinet, Ns-3 y Mininet.

	ESTINET	**NS-3**	**MININET**
Modo de simulación	Si	Si	No
Modo de emulación	Si	No	Si
Compatible con controladores reales	Si	No	Si
Resultado repetible	Si	No	Si
Escalabilidad	Alta (para un solo proceso)	Alta (para un solo proceso)	Media (para múltiples procesos)
Exactitud en los resultados de rendimiento	Si	No admite protocolo de árbol y ningún controlador real	Sin fidelidad de rendimiento
Soporte de GUI	Para configuración y observación	Sólo para observación	Sólo para observación

Fuente: (Valencia et al., 2015)

Los programas que Mininet ejecuta, pueden enviar paquetes por lo que se asemeja mucho a una interfaz real Ethernet, con una velocidad y retardo de enlace determinada. Cuando dos programas, como un cliente y un servidor se comunican utilizando Mininet, el rendimiento medio debe coincidir con el de dos máquinas reales. En conclusión, Mininet ejecuta Switch, Routers, enlaces, controladores, los mismos que se crean utilizando software en lugar de hardware y en su mayoría la forma de comportarse de estos equipos es similar a elementos de hardware reales (Yánez, 2015).

Este emulador ofrece un entorno con línea de comandos simple que permite una interacción amigable del usuario con la red virtualizada; además, cuenta con una API basada

en Python que permite manejar y construir redes de datos a partir de un conjunto de líneas de código, es decir ofrece la construcción de redes personalizadas de acuerdo a nuestros requerimientos (Guerrero, 2017).

4.5.1 Características de Mininet

Bonilla (2016), menciona algunas de las características más relevantes de Mininet.

- **Rapidez:** la creación de una red sencilla tarda unos pocos segundos.
- **Flexibilidad:** permite la interpretación de nuevas topologías y funcionalidades utilizando software basado en lenguajes de programación y Sistemas Operativos comunes.
- **Realista:** permite analizar el tráfico de con un alto grado de confianza ejecutando programas como WireShark.
- **Amigable:** es fácil de usar, se puede experimentar redes sencillas en Mininet escribiendo scripts sencillos.
- **Escalabilidad:** en un solo computador se puede escalar a múltiples elementos de red.
- **Aplicabilidad:** se pueden implementar prototipos en redes basadas en Hardware sin realizar cambios en el código fuente.
- **Compartible:** los prototipos creados en Mininet pueden ser compartidos para que otros desarrolladores puedan utilizarlos y modificarlos si se requiere.
- **Interactividad:** la administración y emulación se realiza en tiempo real.
- **Open Source:** permite examinar, cambiar el código fuente, corregir errores, problemas con los archivos y otros.

4.5.2 Comandos básicos de la Interfaz Mininet

La interfaz de usuario de Mininet es una interfaz por línea de comandos, a través de la cual se generan tolos los dispositivos o elementos necesarios para la emulación de la red (Álvarez, 2017). La tabla 5 muestra los comandos más utilizados dentro de la Interfaz de usuario Mininet.

Tabla 5. Comandos Básicos Interfaz Mininet

COMANDO	FUNCIÓN
$ sudo mn	Ejecutar Mininet
mininet> exit	Salir de Mininet
$ sudo mn -c	Realizar borrado del entorno, de modo que esté listo para volver a utilizarlo.
$ sudo mn -h	Muestra una pequeña ayuda de los diferentes comandos y opciones que soporta mn.
$ sudo mn o **$ sudo mn --topo minimal**	Lo habitual es implementar la topología a la vez que se invoca al entorno mininet. Si no se especifica nada, se implementará la topología "minimal".
$ sudo mn --topo single, [n]	Topología simple donde n es el número de hosts deseados.
$ sudo mn --topo linear, [n]	Topología lineal, donde n es el número de switch.
$ sudo mn --topo tree, depth=n	Topología en árbol, con una profundidad de n niveles. En el nivel n-ésimo cada switch conecta dos hosts.
mininet> nodes	Muestra los nodos disponibles
mininet>net	Muestra la información sobre la red.
mininet>dump	Muestra información sobre las direcciones IP de cada nodo.
mininet> h1 ifconfig -a	ejecutará el comando ifconfig -a desde el host h1.
mininet> link s1 h1 down mininet> link s1 h1 up	Habilitar/inhabilitar un enlace determinado
mininet> h1 ping h2	Ping desde el equipo final h1 al equipo final h2
mininet> pingall	Ping entre todos los equipos finales de red
mininet> iperf h1 h2	Medir ancho de banda entre equipos finales
mininet> xterm h1	Abrir consola de los equipos finales

Fuente: (Roncero, 2016)

4.5.3 Topologías en Mininet

Las topologías disponibles en Mininet son: minimal (mínima), single (única), linear (lineal), tree (árbol), y personalizada. En cualquiera de estas existe un controlador, varían en el número de host, Switch y los enlaces entre estos (Valencia et al., 2015).

4.5.3.1 *Topología minimal*

Si no se especifica nada, se implementará la topología "minimal". Esto es:

$ sudo mn o **$ sudo mn --topo minimal**

L a figura 14 muestra una topología mínima.

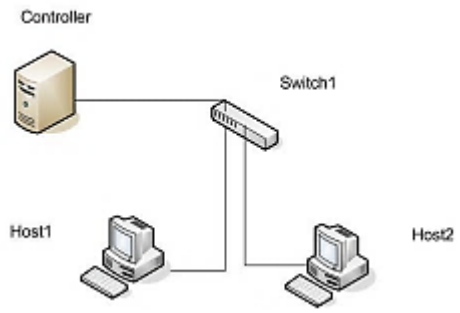


Figura 14. Topología Mínima

Fuente: (Roncero, 2016)

Esta topología consta de un controlador (si no se especifica, se selecciona el controlador por defecto que trae OpenFlow), un Switch OpenFlow y dos hosts conectados al mismo (Roncero, 2016).

4.5.3.2 *Topología single (única)*

Esta topología se crea mediante el comando:

$ sudo mn --topo single, [n]

Donde **n** es el número de host conectados a un único Switch (ver figura 15).

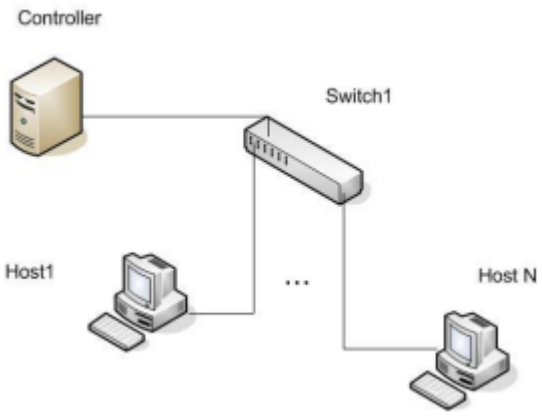


Figura 15. Topología Single.

Fuente: (Roncero, 2016)

4.5.3.3 *Topología linear*

Esta topología se crea mediante el comando:

$ sudo mn --topo linear, [n]

En esta topología **n** es el número de Switch. Consta de n Switches conectados entre sí. En cada Switch, se conecta un host (Roncero, 2016). Esta topología se muestra en la figura 16.

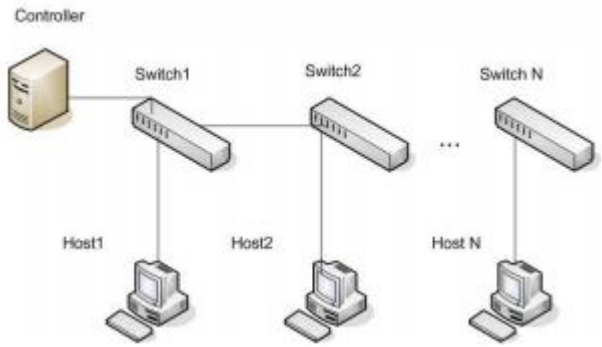


Figura 16. Topología Linear

Fuente: (Roncero, 2016)

4.5.3.4 *Topología Tree*

Esta topología se crea con el comando:

$ sudo mn --topo tree, depht=n, fanout =m

Crea una topología de árbol, con profundidad de **n** niveles y **m** host conectados a cada Switch (ver figura 17). El nivel superior es de un Switch, y cada Switch se conecta a dos más en el nivel inferior, en el n-ésimo nivel, cada Switch conecta dos hosts (Roncero, 2016).

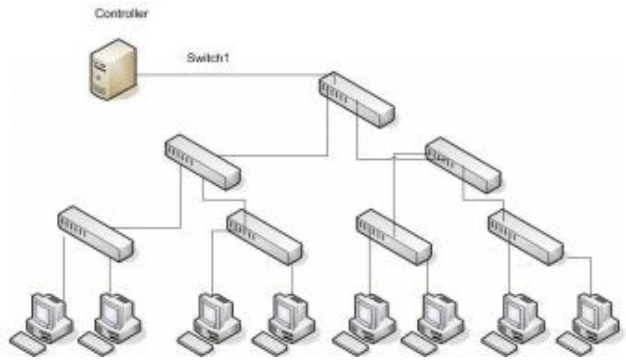


Figura 17. Topología Tree

Fuente: (Roncero, 2016)

4.5.3.5 *Topología personalizada (custom)*

Se pueden implementar topologías tan complejas como se desee, para ello se crea lo que se conoce como topologías customatizadas a través de un script basado en lenguaje *Python* (España, 2016). La figura 18 muestra un ejemplo de una topología personalizada, formada por dos Switches interconectados y dos hosts en cada Switch.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost1 = self.addHost( 'h1' )
        rightHost1 = self.addHost( 'h2' )

        leftHost2 = self.addHost ('h3')
        rightHost2 = self.addHost ('h4')

        leftSwitch = self.addSwitch( 's1' )
        rightSwitch = self.addSwitch( 's2' )

        # Add links
        self.addLink( leftHost1, leftSwitch )
        self.addLink( rightHost1,leftSwitch)

        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, leftHost2)
        self.addLink( rightSwitch, rightHost2 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figura 18. Topología Personalizada

Fuente: (Roncero, 2016)

La tabla 6 muestra algunos de los comandos más importantes utilizados para crear un script Python y poder emular una topología de red personalizada.

Tabla 6. Comandos Básicos API Mininet Python

Comando	Funcionalidad
CLI	Permite la generación de una interfaz de usuario al iniciar la script
Intf	Permite caracterizar una interfaz de forma genérica
TCIntf	Permite caracterizar un TC en la interfaz
Link	Genera un enlace entre dos nodos
TCLink	Genera un TC en el enlace entre dos nodos cualesquiera
TCULink	Genera un TC optimizado para enlaces entre nodos del tipo Userspace
MiniNet	Clase creadora de la red emulada
Host	Genera un host como nodo de red
UserSwitch	Genera un switch del tipo User-space
OVSSwitch	Genera un switch del tipo OVS
RemoteController	Permite indicar a MiniNet la existencia de un controlador externo al entorno de emulación

Fuente: (Álvarez, 2017)

4.6 Switch SDN Zodiac FX

Zodiac FX está diseñado como una placa de implementación de red de cuatro puertos que está desarrollada para investigadores y desarrolladores para modelar y diseñar servicios y aplicaciones SDN. Se considera el conmutador más pequeño: lo suficientemente flexible como para adaptarse a los usuarios de escritorio (Nazir, Humayun, Ahmad & Elías, 2017).

Proporciona un código de firmware que da al usuario flexibilidad para generar varias versiones propias. Proporciona varios tipos de dispositivos, como las aplicaciones que

pueden tener enrutador, puente, equilibrador de carga, servidor web, concentrador VPN, cliente TOR. La línea de comando Zodiac FX está disponible a través del puerto USB (Nazir, et al., 2017).

Para configurar Zodiac FX se utiliza Software como Minicom, Putty, gtkterm, según menciona Nazir, et al. (2017):

La Figura 19 muestra el puerto para Zodiac Fx.

Figura 19. Conexión al Puerto Zodiac FX

Fuente: (Nazir, et al., 2017)

El software de configuración proporciona la forma de configurar el Zodiac FX y proporciona el puerto para configurar. Una vez elegido el puerto, se puede configurar Zodiac FX, esto se muestra en la figura 20. Como se menciona en el manual Zodiac FX, el puerto cuatro es el puerto del plano de control OpenFlow, mientras que otros puertos son los puertos del plano de reenvío OpenFlow (Nazir, et al., 2017).

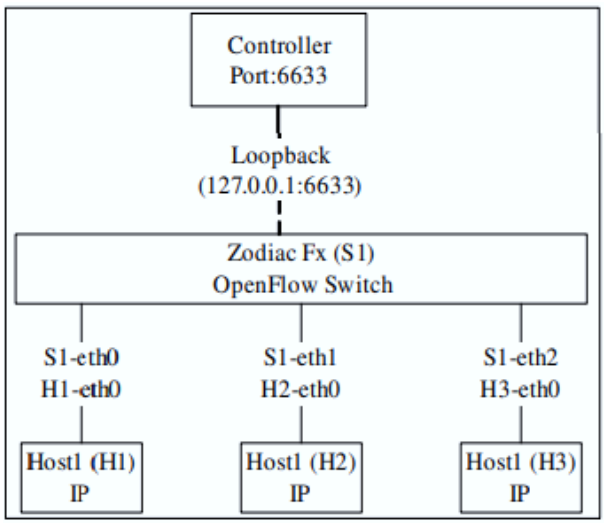


Figura 20. Conexión del Switch Zodiac FX con el Controlador Ryu.

Fuente: (Nazir, et al., 2017)

Los componentes físicos y algunas características adicionales que corresponden al Switch Zodiac FX según (Networks, Kickstarter, 2017), se describen a continuación y se muestran en la Figura 21.

- ✓ 4 puertos Fast Ethernet 10/100.
- ✓ Interfaz de línea de comando accesible a través del puerto serie virtual USB.
- ✓ Procesador Amtel ATSAM4E Cortex M4
- ✓ Soporte para OpenFlow 1.0, 1.3 y 1.4
- ✓ Tabla de flujo de software de 512 entradas.
- ✓ Autenticación por puerto basado en 802.1x.
- ✓ Protocolo de árbol de expansión rápida 802.1w (RSTP)
- ✓ 16 ACL por puerto
- ✓ Priorización de QoS / CoS con inserción de etiqueta 802.1q
- ✓ LED de actividad
- ✓ Alimentación del dispositivo mediante USB
- ✓ Tamaño pequeño de tan solo 10cm x 8 cm.
- ✓ Cabecera de expansión SPI de alta velocidad.

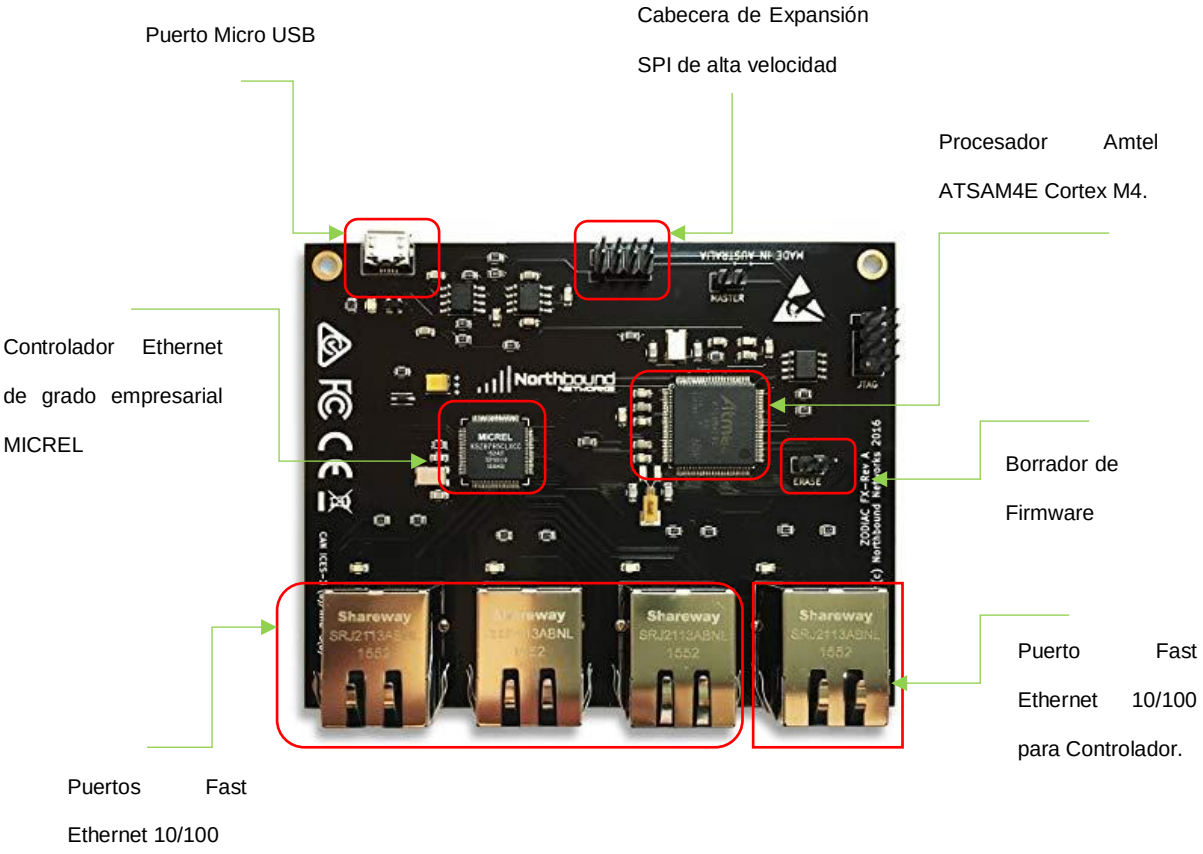


Figura 21. Componentes Switch Zodiac FX.

(Networks, Kickstarter, 2017)

5 MATERIALES Y METODOS

5.1 Materiales

De acuerdo al área de aplicación para el desarrollo del proyecto antes mencionado, se utilizarán los materiales enunciados a continuación:

5.1.1 Materiales de Hardware

Tabla 7. Materiales se Hardware

Material	Características	Función
Computador	Portátil	Portador del Controlador RYU
Switch Zodiac FX		Dispositivo físico para elaboración de Prácticas
Patch Cord	Cat. 6	Conexión de Switch Zodiac FX hacia computador
Materiales de oficina	n/a	n/a

Fuente: (El autor)

5.1.2 Materiales de Software

Tabla 8. Materiales de Software

Material	Función
Ubuntu GNU/Linux	Sistema Operativo base sobre el cual opera el Controlador Ryu.
Mininet	Simulador de Redes de Datos.
RYU	Controlador SDN

Python	Lenguaje de Programación para creación de Scripts.
MiniEdit	Editor de interfaz gráfica de usuario (GUI) para Mininet.
Wireshark	Herramienta de Software para capturar el tráfico de paquetes.
Curl	Herramienta para hacer peticiones http
Putty	Emulador de terminal gratuito.

Fuente: (El autor)

5.1.3 Presupuesto

El presupuesto total para la Implementación del Proyecto se resume en la siguiente tabla:

Tabla 9. Presupuesto

Cantidad	**Material**	**Precio**
1	Switch Zodiac FX	$119
1	Computador Portátil Toshiba Satélite Corei7	$800
4	Patch Cord	$8.00
	TOTAL=	**$927**

Fuente: (El autor)

5.2 Métodos

5.2.1 Método Analítico

Permitirá realizar un análisis de las partes que conforman el diseño, en este caso del Switch Zodiac FX y del Controlador Ryu, este análisis se lo realizará de acuerdo a las configuraciones que se den sobre el Zodiac FX y se observará su rendimiento.

5.2.2 Método Científico

Este método brindara la facilidad de realizar un registro de los acontecimientos y estudios realizados previamente al desarrollo del siguiente proyecto, y fundamentarlos con los resultados que se presenten en la implementación del sistema para un análisis y formulación de conclusiones y recomendaciones.

5.2.3 Método Experimental

Permitirá a través del desarrollo de prácticas tanto virtuales utilizando el Simulador Mininet, y de pruebas físicas con el Switch Zodiac SDN, establecer los fundamentos para el desarrollo de futuros trabajos relacionados con SDN.

5.2.4 Método Comparativo

Permitirá establecer una comparativa entre la Solución planteada en el proyecto y soluciones tradicionales, para establecer ventajas, conclusiones y recomendaciones.

5.3 Desarrollo del proyecto

Una vez descrita la metodología, se procede con la instalación de los diferentes escenarios de configuración y pruebas.

5.3.1 Sistema Operativo Ubuntu GNU/Linux

Para el desarrollo del presente Proyecto se utiliza el Sistema Operativo Ubuntu GNU/Linux 16.04.

5.3.2 Instalación de Mininet

Para la instalación de Mininet se obtiene el código desde el repositorio y lo clonaremos utilizando el comando git clone desde la consola de Ubuntu en modo root. Este proceso se muestra en la figura 22.

```
root@mininetryu:~# git clone http://github.com/mininet/mininet
Clonar en «mininet»...
remote: Enumerating objects: 9677, done.
remote: Total 9677 (delta 0), reused 0 (delta 0), pack-reused 9677
Receiving objects: 100% (9677/9677), 3.00 MiB | 189.00 KiB/s, done.
Resolving deltas: 100% (6425/6425), done.
Comprobando la conectividad… hecho.
root@mininetryu:~#
```

Figura 22. Comando para obtener el código fuente de Mininet desde Repositorio.

Fuente: (El autor)

Se puede mostrar la lista de tags del repositorio y hacer un checkout de la versión 2.2.2 de Mininet. Esto se evidencia en la siguiente figura.

```
root@mininetryu:~# git tag
fatal: Not a git repository (or any of the parent directories): .git
root@mininetryu:~# cd mininet/
root@mininetryu:~/mininet# git tag
1.0.0
2.0.0
2.1.0
2.1.0p1
2.1.0p2
2.2.0
2.2.1
2.2.2
2.3.0d3
2.3.0d4
2.3.0d5
2.3.0d6
cs244-spring-2012-final
root@mininetryu:~/mininet# git checkout -b 2.2.2
Switched to a new branch '2.2.2'
root@mininetryu:~/mininet#
```

Figura 23. Comando que muestra la lista de tags y checkout de la versión 2.2.2

Fuente: (El autor)

Se procede a utilizar el script de instalación install.sh para completar este proceso (ver figura 24). La opción “nfv”, instala además de Mininet el protocolo OpenFlow y Open vSwitch. La salida que muestra el comando de instalación es bastante extensa, por lo que se muestra únicamente algunas líneas.

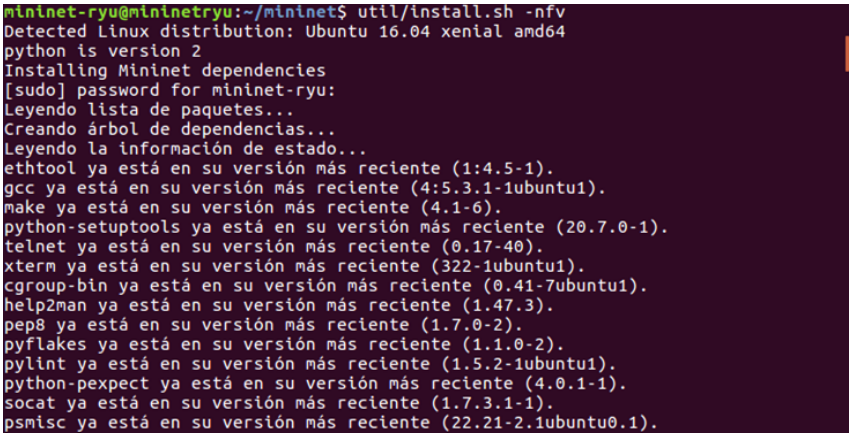


Figura 24. Comando de Instalación de Mininet, OpenFlow y Open vSwitch

Fuente: (El autor)

Si se presenta algún error al momento de realizar la instalación, se debe ingresar al archivo install.sh y actualizar los repositorios; ya que, al tratarse de Software libre, este se mantiene actualizándose constantemente y puede existir algún paquete obsoleto que nos de algún tipo de error.

5.3.3 Prueba de Funcionamiento de Mininet

Una vez instalado Mininet se puede realizar pruebas de funcionamiento en el mismo. Antes de realizar cualquier proceso dentro del simulador es recomendable limpiar la consola utilizando el comando ***sudo mn -c***. Este paso se muestra en la figura siguiente:

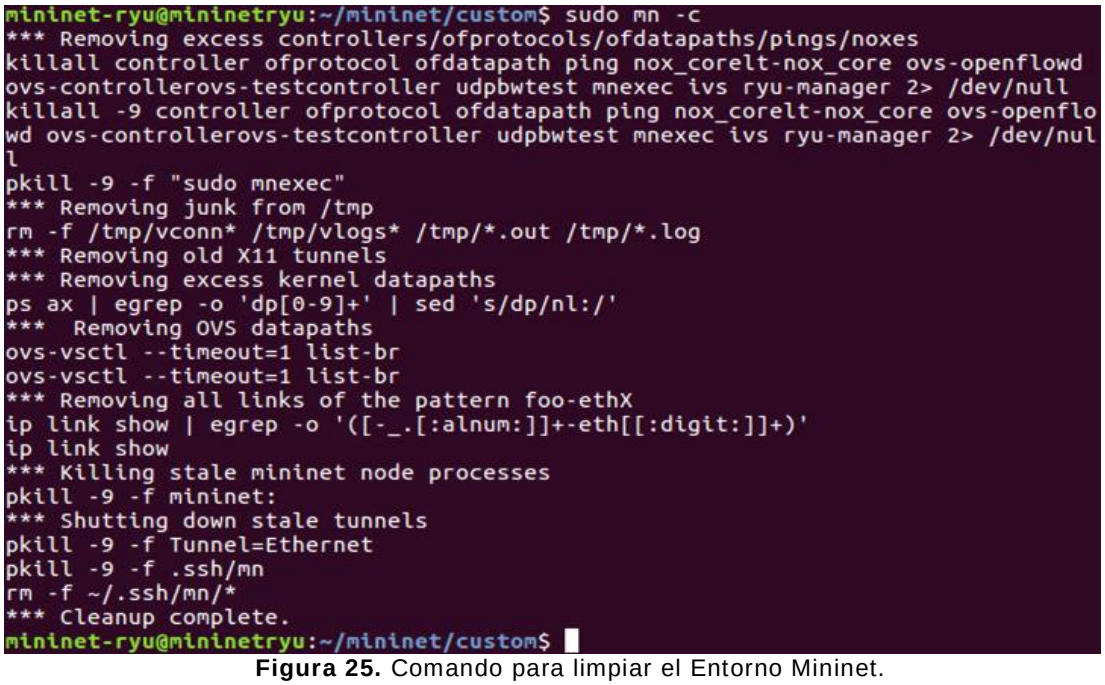


Figura 25. Comando para limpiar el Entorno Mininet.

Fuente: (El autor)

De esta forma ya podemos arrancar Mininet, utilizando el comando ***sudo mn*** que nos muestra una topología simple de un Switch y dos hosts.

```
mininet-ryu@mininetryu:~/mininet/custom$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figura 26. comando para iniciar Mininet con una Topología Simple

Fuente: (El autor)

5.3.4 Instalación de Ryu

Antes de instalar el controlador Ryu es necesario instalar algunas herramientas adicionales, para ello utilizamos el siguiente comando:

```
mininet-ryu@mininetryu:~$ sudo apt-get -y install git python-pip python-dev
```

También es necesario instalar algunos paquetes de Python utilizando el siguiente comando:

```
mininet-ryu@mininetryu:~$ sudo apt-get -y install python-eventlet python-routes
python-webob python-paramiko
```

Existen dos formas de instalar Ryu, una forma directa utilizando el comando ***pip*** y la otra forma utilizando el repositorio e instalando manualmente. En nuestro caso utilizaremos la segunda opción que es clonando desde el repositorio.

Instalación Directa

Se utiliza el comando siguiente:

```
mininet-ryu@mininetryu:~$ pip install ryu
```

Instalación Manual

Paso 1. Para clonar Ryu desde el repositorio se utiliza el siguiente comando:

```
mininet-ryu@mininetryu:~$ git clone --depth=1 https://github.com/osrg/ryu.git
```

Paso 2. Actualizamos los repositorios

```
mininet-ryu@mininetryu:~$ sudo pip install setuptools --upgrade
```

Paso 3. Procedemos a la instalación desde el directorio */ryu* llamando el archivo de instalación *setup.py*:

```
mininet-ryu@mininetryu:~/ryu$ sudo python ./setup.py install
```

Paso 4. Si se producen errores al omento de ejecutar la instalación, puede que falten módulos complementarios, por lo que a continuación se detallan algunos comandos que serán útiles en caso de existir algún error. Además, permitirán actualizar algunos paquetes de Python.

```
mininet-ryu@mininetryu:~/ryu$ sudo pip install oslo.config msgpack-python
mininet-ryu@mininetryu:~/ryu$ sudo pip install six --upgrade
mininet-ryu@mininetryu:~/ryu$ sudo pip install eventlet --upgrade
```

Paso 5. Se puede además observar la versión de ***ryu-manager*** que servirá para ejecutar los archivos ***.py.***

```
mininet-ryu@mininetryu:~/ryu$ ryu-manager --version
ryu-manager 3.30
```

Paso 6. Se puede encontrar todas las aplicaciones que Ryu trae consigo ingresando al directorio ***ryu/ryu/app***, esto se evidencia en la figura 27.

```
mininet-ryu@mininetryu:~/ryu$ ls ryu/app/
bmpstation.py          rest_router.py            simple_switch_lacp.py
cbench.py              rest_topology.py          simple_switch.py
conf_switch_key.py     rest_vtep.py              simple_switch_rest_13.py
example_switch_13.py   simple_monitor_13.py      simple_switch_snort.py
gui_topology           simple_switch_12.py       simple_switch_stp_13.py
__init__.py            simple_switch_13.py       simple_switch_stp.py
ofctl                  simple_switch_14.py       simple_switch_websocket_13.py
ofctl_rest.py          simple_switch_15.py       wsgi.py
rest_conf_switch.py    simple_switch_igmp_13.py  ws_topology.py
rest_firewall.py       simple_switch_igmp.py
rest_qos.py            simple_switch_lacp_13.py
```

Figura 27. Aplicaciones Incluidas en Ryu.

Fuente: (El autor)

5.3.5 Instalación de Wireshark

Wireshark es una herramienta que nos servirá para el análisis de trafico de las diferentes prácticas que se desarrolla en este Proyecto, antes de proceder con su instalación es necesario actualizar la lista de paquetes disponibles y sus versiones; además, se añade el repositorio antes de realizar la instalación. Este proceso se muestra en la figura 28.

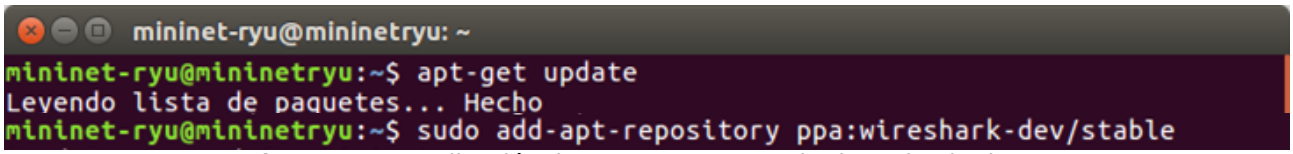


Figura 28. Actualización de paquetes y Repositorios Wireshark.

Fuente: (El autor)

Una vez actualizados los paquetes y repositorios, se procede con la instalación de Wireshark con la ayuda del comando que aparece en la figura 29.

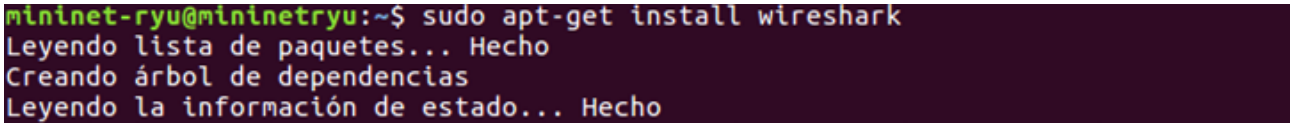


Figura 29. Comando de Instalación Wireshark.

Fuente: (El autor)

5.3.6 Instalación de cURL

Esta herramienta nos sirve para gestionar lenguajes de programación, en nuestro caso Python. Además, sirve para gestionar peticiones http.

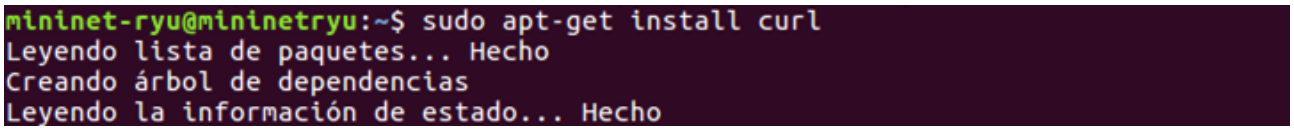


Figura 30. Instalación de cURL.

Fuente: (El autor)

5.3.7 Instalación de SAM-BA

SAM-BA es un programa con infinitas utilidades, pero en nuestro Proyecto nos permitirá la actualización de firmware del Zodiac FX de forma rápida y eficaz.

Paso 1. Ejecutamos el archivo .exe y pulsamos next (ver figura 31)

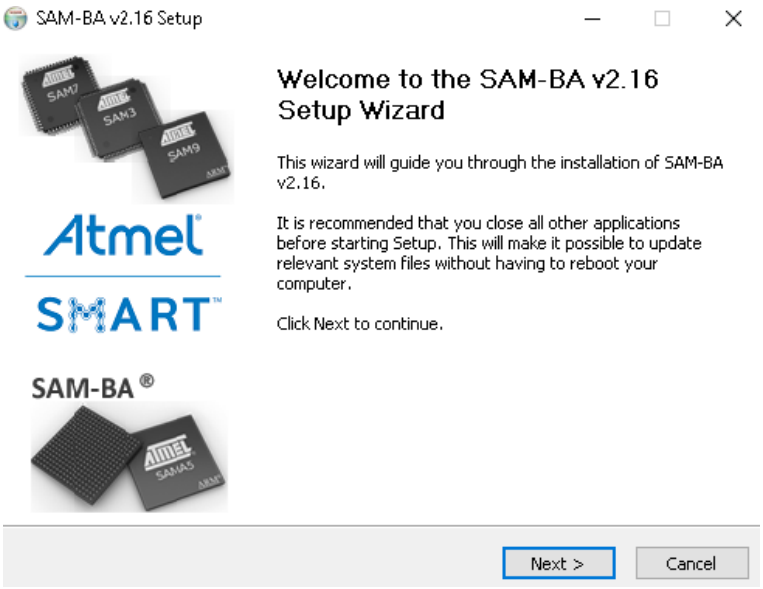


Figura 31. Inicio de Instalación SAM-BA

Fuente: (El autor)

Paso 2. Aceptamos términos y condiciones y agregamos licencia (ver figura 32).

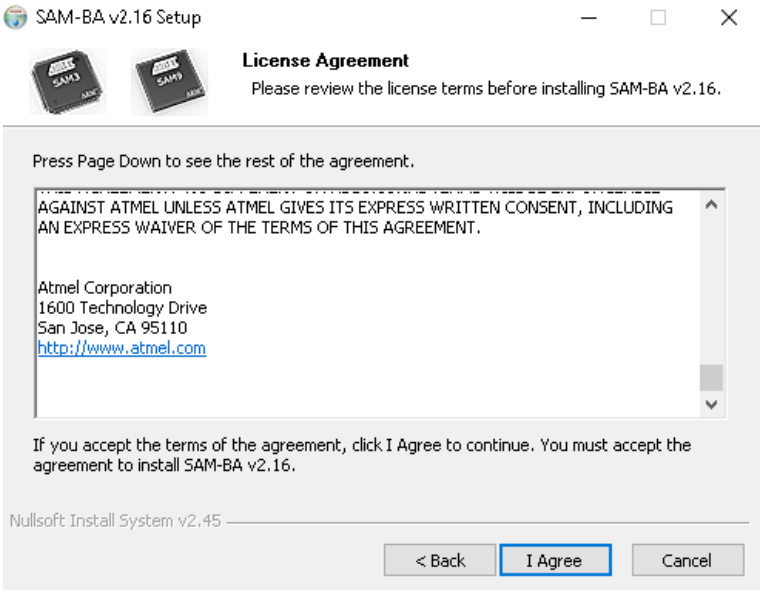


Figura 32. Términos, Condiciones y Licencia.

Fuente: (El autor)

Paso 3. Se muestra la versión de SAM-BA que vamos a Instalar (ver figura 33).

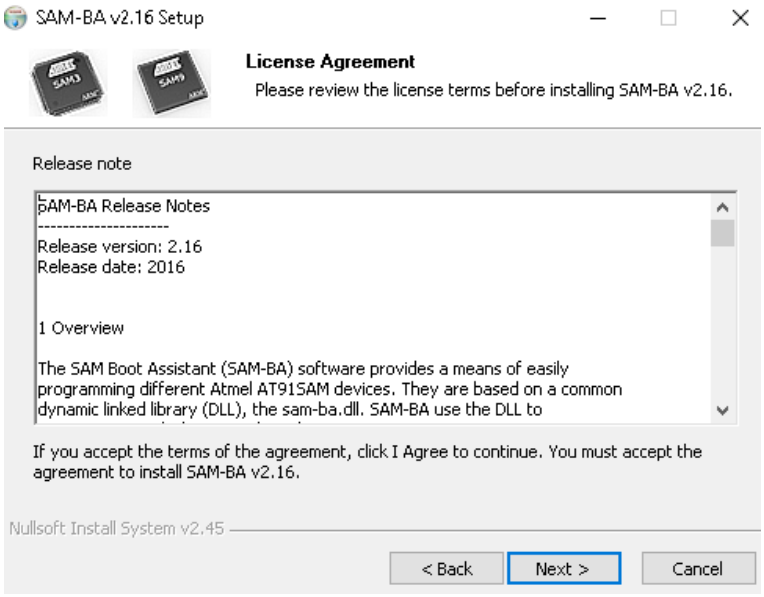


Figura 33. Versión de SAM-BA.

Fuente: (El autor)

Paso 4. Seleccionamos la carpeta de destino y procedemos con la Instalación (ver figura 34 y 35).

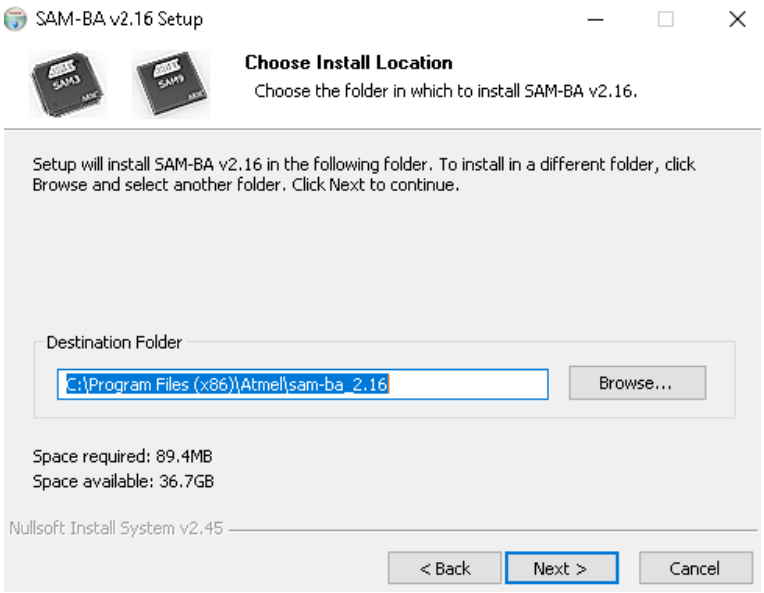


Figura 34. Carpeta de Destino de SAM-BA.

Fuente: (El autor)

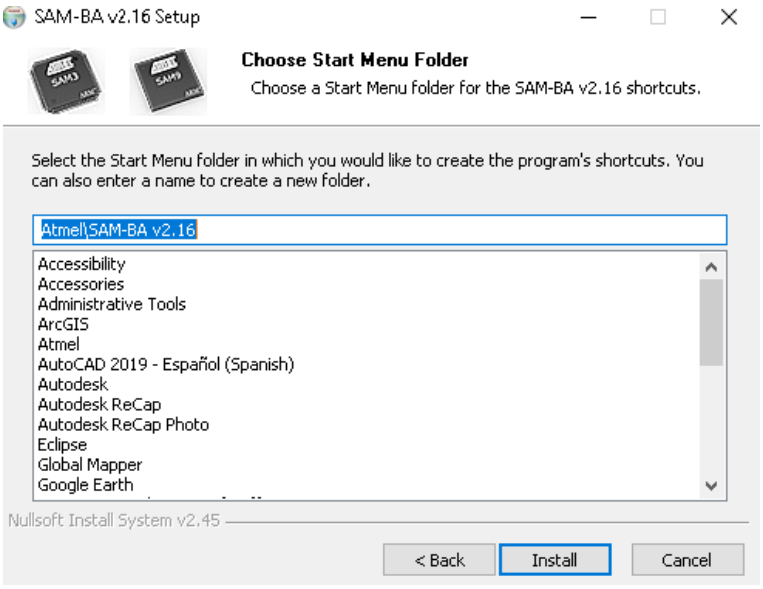


Figura 35. Instalación de SAM-BA.

Fuente: (El autor)

Paso 4. Una vez finalizada la instalación se nos muestra un manual de Usuario y tenemos nuestro programa listo para utilizarlo (ver figura 36 y 37).

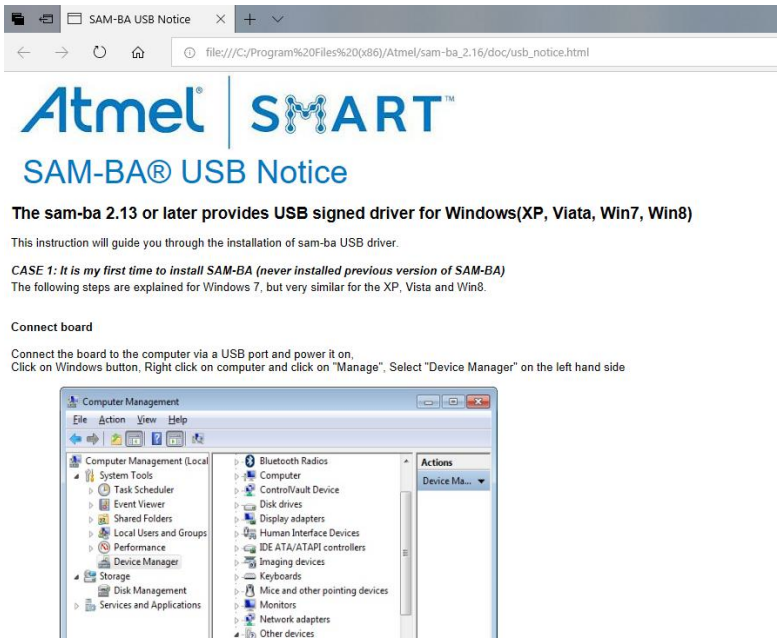


Figura 36. Manual de Usuario SAM-BA.

Fuente: (El autor)

Figura 37. Ejecución de SAM-BA.

Fuente: (El autor)

5.3.8 Instalación de Putty

Putty es un emulador de terminal gratuito que nos ayudara a realizar configuraciones y ver el estado del Zodiac FX, su instalación es muy sencilla y se hace utilizando el comando que aparece en la figura 38.

```
mininet-ryu@mininetryu:~$ apt-get update
Leyendo lista de paquetes... Hecho
mininet-ryu@mininetryu:~$ sudo apt-get install putty
[sudo] password for mininet-ryu:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

Figura 38. Instalación de Emulador Putty

Fuente: (El autor)

.

6 RESULTADOS

Para llevar a cabo el desarrollo de las Practicas, tanto en el simulador Mininet como en el Switch físico Zodiac FX se plantea un formato de Practicas que se debe seguir:

Título de la Práctica

Objetivos:

Recursos utilizados:

Recursos de Hardware	**Recursos de Software**

Procedimiento:

Resultados:

Conclusiones:

Recomendaciones:

6.1 Resultados usando el simulador Mininet

En esta sección se plantea cinco practicas utilizando el simulador Mininet. Para el desarrollo de las prácticas 1, 2 y 3 nos basamos en el estudio hecho según (Lamallam, 2017), y las practicas 4 y 5 son de autoría propia.

6.1.1 Practica 1:

Instalación y configuración de Mininet y RYU

Objetivos:

- Crear el entorno de trabajo para el desarrollo de Practicas, a través de la instalación de Mininet, Ryu y herramientas complementarias.
- Entender el funcionamiento de la herramienta de simulación Mininet y del Controlador SDN Ryu, así como de las herramientas complementarias utilizadas para llevar a cabo el desarrollo de las prácticas.

Recursos utilizados:

Tabla 10. Recursos Práctica 1 Mininet

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Acceso a Internet (Mozilla Firefox) - Centro de Software de Ubuntu.

Fuente: (el autor)

Procedimiento:

El procedimiento llevado a cabo para la instalación del Simulador Mininet, el Controlador Ryu y herramientas de Software adicionales se detallan secciones anteriores como se indica a continuación:

4.3.2 Instalación de Mininet

4.3.3 Funcionamiento de Mininet

4.3.4 Instalación de Ryu

4.3.5 Instalación de Wireshark

4.3.6 Instalación de cURL

4.3.7 Instalación de SAM-BA

4.3.8 Instalación de Putty

Resultados:

Los resultados de la Instalación de Mininet, Ryu y herramientas complementarias para el desarrollo de las practicas propuestas, así como el funcionamiento de las mismas se detallan de igual forma en las secciones descritas con anterioridad en Procedimiento.

Conclusiones:

- Se instaló la herramienta de Simulación Mininet, el controlador SDN Ryu y herramientas de Software adicionales para el desarrollo de prácticas propuestas.
- Se logró comprender el funcionamiento del simulador Mininet, el controlador Ryu y las Aplicaciones que trae consigo, así como el uso de herramientas complementarias.

Recomendaciones:

- Se recomienda utilizar repositorios oficiales para la instalación de las herramientas de software descritas anteriormente con la finalidad de obtener información actualizada de las mismas.
- Es recomendable entender el funcionamiento de las aplicaciones que proporciona el controlador Ryu para utilizarlas en prácticas posteriores.

6.1.2 Practica 2:

Aplicación Firewall configurado por interfaz REST

Objetivos:

- Utilizar el simulador Mininet para desarrollar la topología de red., Ryu como controlador SDN, REST como protocolo de NorthBound y OpenFlow como protocolo de SouthBound.
- Proporcionar a los Switches reglas para que cumplan funciones de Firewall, y a través de Ryu arrancar la aplicación configurada mediante interfaz REST.

Recursos utilizados:

Tabla 11. Recursos Practica 2 Mininet

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Simulador Mininet - Controlador Ryu - OpenFlow v1.3 - Aplicación Python

Fuente: (El autor)

Procedimiento:

La topología utilizada para esta práctica se muestra en la figura 39.

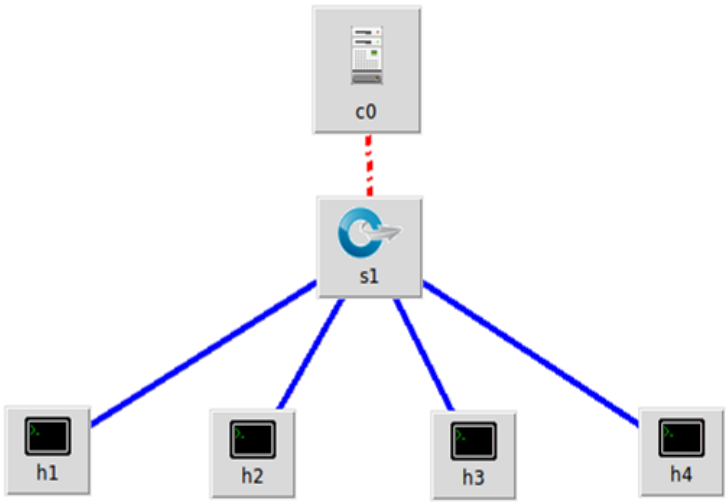


Figura 39. Topología para la Practica 2 (Firewall REST).

Fuente: (El autor)

Así mismo la figura 40, muestra el script que define la red de la figura anterior.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def _init_( self ):
        "Create custom topo."
        # Initialize topology
        Topo._init_( self )


        # Add hosts and switches
        h1Host = self.addHost( 'h1' )
        h2Host = self.addHost( 'h2' )
        h3Host = self.addHost( 'h3' )
        h4Host = self.addHost( 'h4' )
        s1Switch = self.addSwitch( 's1' )

        # Add links
        self.addLink( h1Host, s1Switch )
        self.addLink( h2Host, s1Switch )
        self.addLink( h3Host, s1Switch )
        self.addLink( h4Host, s1Switch )


topos = { 'mytopo2': ( lambda: MyTopo() ) }
```

Figura 40. Código en Python para la Topología de la Practica 2.

Fuente: (El autor)

- **Arranque de Mininet**

  Arrancamos Mininet llamando la topología que hemos proporcionado en la ruta que ésta se encuentre, e indicaremos que se usará un controlador remoto que se ejecuta dentro de la misma máquina virtual. El resultado se muestra en la figura 41.

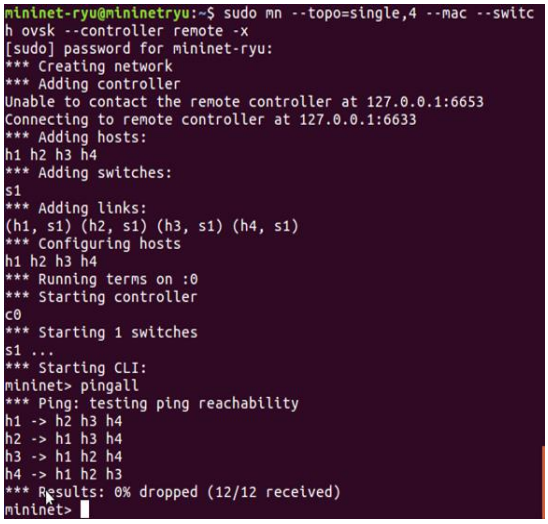


Figura 41. Arranque de la Topología en Mininet para la Practica 2.

Fuente: (El autor)

Este comando además despliega una interfaz para cada uno de los elementos de la red, esto se aprecia en la figura 42. Este resultado se aprecia gracias al uso del controlador remoto, y consta de un controlador c0, un Switch s1 y tres host h1, h2 y h3.

Figura 42. Despliegue de los elementos de Red utilizando un controlador Remoto.

Fuente: (El autor)

- **Arranque de Ryu**

Para el arranque de Ryu se utiliza el comando *ryu-manager* seguido del nombre de la aplicación, en este caso *rest_firewall.py.* Esto se evidencia en la figura 43.

Figura 43. Comando Ryu para arrancar la Aplicación Firewall REST

Fuente: (El autor)

- **Configuración Comandos REST**

Es necesario habilitar el firewall del Switch en el cual se está trabajando, en este caso el Switch 1, para ello se utiliza el siguiente comando.

curl -X PUT http://localhost:8080/firewall/module/enable/0000000000000001

Para establecer la comunicación entre los elementos de red usaremos los comandos REST que se muestran a continuación.

Habilitamos tráfico ICMP entre host 1 y host2

curl -X POST -d '{"nw_src": "10.0.0.1/32", "nw_dst": "10.0.0.2/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.1/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

Habilitamos tráfico ICMP entre host 1 y host3

curl -X POST -d '{"nw_src": "10.0.0.1/32", "nw_dst": "10.0.0.3/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst": "10.0.0.1/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

Habilitamos tráfico ICMP entre host 2 y host4

curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.4/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

curl -X POST -d '{"nw_src": "10.0.0.4/32", "nw_dst": "10.0.0.2/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

Resultados:

Se puede observar toda la configuración del Switch utilizando el siguiente comando:

Figura 44. Comando REST para ver todas las configuraciones del Switch.

Fuente: (El autor)

Se puede apreciar, además los flujos de entrada en el Switch luego de añadir todas las reglas a través de la interfaz REST.

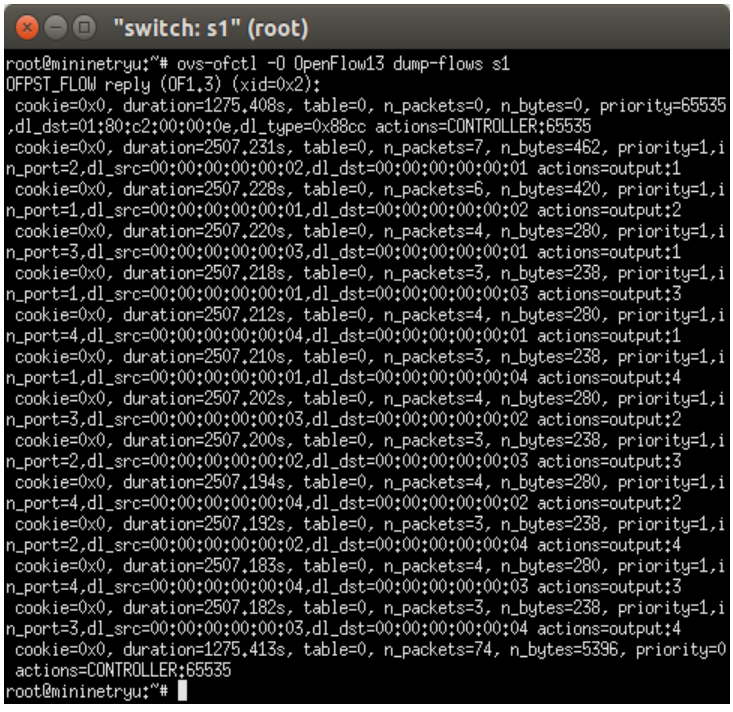


Figura 45. Flujos de entrada en el s1.

Fuente: (El autor)

Se ejecuta el comando pingall dentro de la consola de Mininet para verificar que todos los elementos de red puedan comunicarse. (ver figura 46).

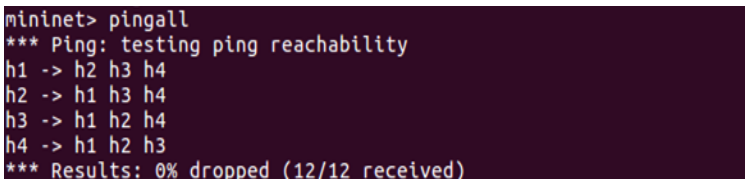


Figura 46. Salida del comando pingall.

Fuente: (El autor)

Se puede apreciar la tabla de flujos del Switch dentro de la consola de Mininet a través del comando que se muestra en la figura 47.

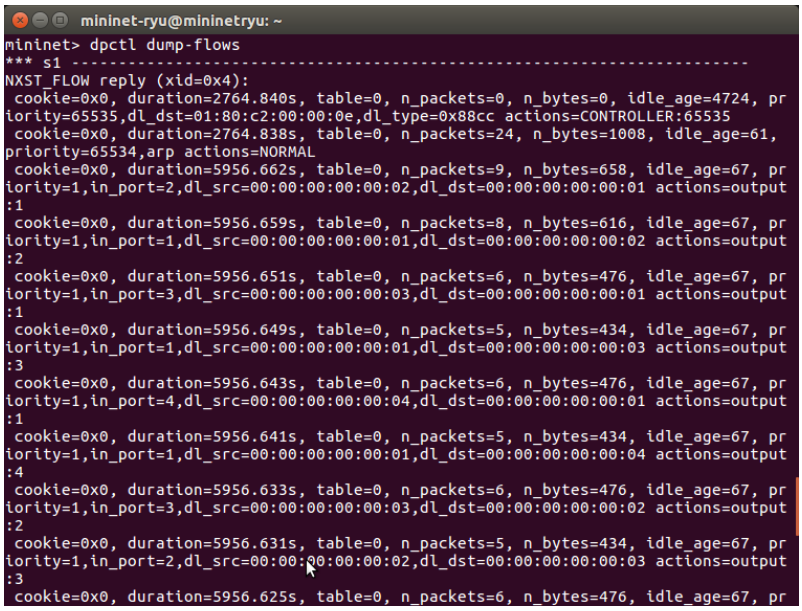


Figura 47. Comando Mininet que muestra las Tablas de flujo del Switch.

Fuente: (El autor)

Establecemos un ping a través del protocolo ICMP activado durante la configuración de reglas, el resultado se aprecia en la figura 48.

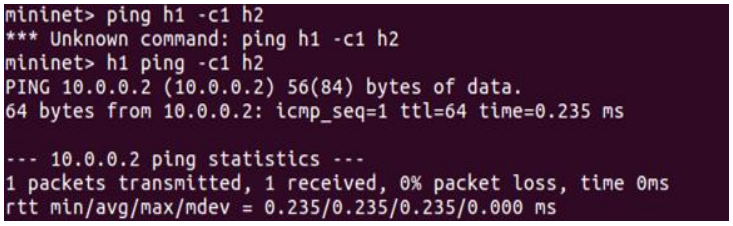
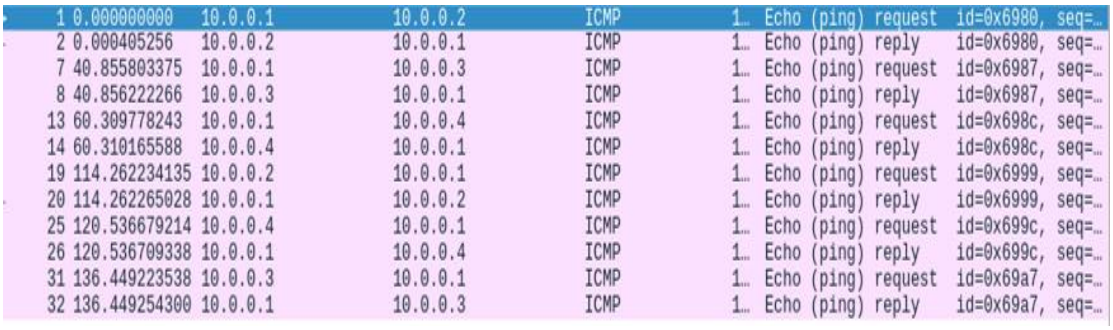


Figura 48. Ping entre h1 y h2.

Fuente: (El autor)

Se utiliza también la herramienta WireShark para ver el resultado de conectividad entre h1 y h2 luego de establecer el ping.



1	0.000000000	10.0.0.1	10.0.0.2	ICMP	1…	Echo (ping) request id=0x6980, seq=…
2	0.000405256	10.0.0.2	10.0.0.1	ICMP	1…	Echo (ping) reply id=0x6980, seq=…
7	40.855803375	10.0.0.1	10.0.0.3	ICMP	1…	Echo (ping) request id=0x6987, seq=…
8	40.856222266	10.0.0.3	10.0.0.1	ICMP	1…	Echo (ping) reply id=0x6987, seq=…
13	60.309778243	10.0.0.1	10.0.0.4	ICMP	1…	Echo (ping) request id=0x698c, seq=…
14	60.310165588	10.0.0.4	10.0.0.1	ICMP	1…	Echo (ping) reply id=0x698c, seq=…
19	114.262234135	10.0.0.2	10.0.0.1	ICMP	1…	Echo (ping) request id=0x6999, seq=…
20	114.262265028	10.0.0.1	10.0.0.2	ICMP	1…	Echo (ping) reply id=0x6999, seq=…
25	120.536679214	10.0.0.4	10.0.0.1	ICMP	1…	Echo (ping) request id=0x699c, seq=…
26	120.536709338	10.0.0.1	10.0.0.4	ICMP	1…	Echo (ping) reply id=0x699c, seq=…
31	136.449223538	10.0.0.3	10.0.0.1	ICMP	1…	Echo (ping) request id=0x69a7, seq=…
32	136.449254300	10.0.0.1	10.0.0.3	ICMP	1…	Echo (ping) reply id=0x69a7, seq=…

Figura 49. Resultado de prueba de eco entre h1 y h2 utilizando la herramienta Wireshark.

Fuente: (El autor)

Conclusiones:

- Se utilizó el simulador Mininet para establecer la topología de red, Ryu como controlador SDN, REST como protocolo de NorthBound y OpenFlow como protocolo de SouthBound.
- Se configuró reglas en el Switch para que cumpla funciones de Firewall, y a través de Ryu arrancar la aplicación configurada mediante interfaz REST.

Recomendaciones:

- Se recomienda utilizar una topología base proporcionada por Mininet, o en su defecto una topología personalizada creada a través de la herramienta Miniedit que ofrece Mininet.
- Es recomendable utilizar la herramienta Wireshark para observar el comportamiento del tráfico generado entre usuarios o hosts dentro de la red creada.

6.1.3 Practica 3:

Aplicación Router configurado por Interfaz REST.

Objetivos:

- Utilizar el simulador Mininet para crear r la topología de red, Ryu como controlador SDN, REST como protocolo de NorthBound y OpenFlow como protocolo de SouthBound.
- Configurar un Switch para que cumpla funciones de Router y arrancar la aplicación a través de Ryu, además de configurar las direcciones IP de las interfaces utilizadas.

Recursos utilizados:

Tabla 12. Recursos Practica 3 Mininet

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Simulador Mininet - Controlador Ryu - OpenFlow v1.3 - Aplicación Python

Fuente: (El autor)

Procedimiento:

La topología utilizada para esta práctica la denominaremos *ruteo.py,* y define la red que se muestra en la figura 50.

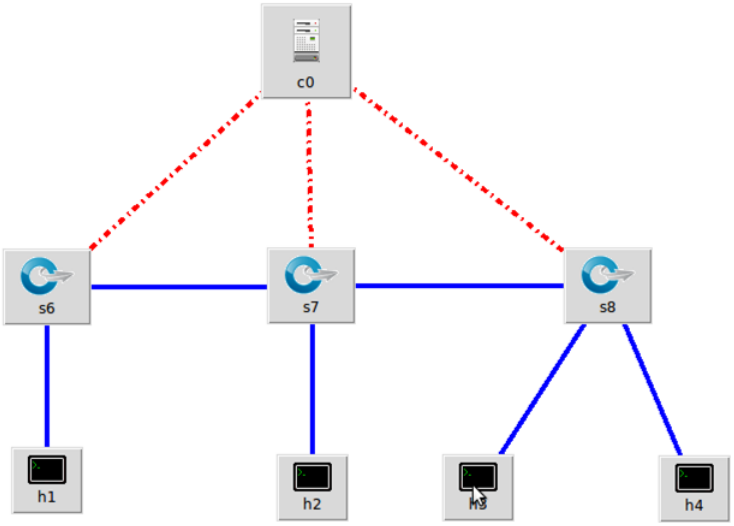


Figura 50. Topología de Mininet para Práctica 3.

Fuente: (El autor)

El código en Python utilizado para esta Práctica se muestra en el **Anexo 1** de este documento.

-Arranque de RYU

De igual forma que en la práctica anterior arrancamos la aplicación *rest_router.py* utilizando el comando que muestra la figura 51.

```
mininet-ryu@minetryu:~/ryu/ryu/app$ ryu-manager rest_router.py
loading app rest_router.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app rest_router.py of RestRouterAPI
(6022) wsgi starting up on http://0.0.0.0:8080
```

Figura 51. Comando Ryu para arrancar la Aplicación Router REST

Fuente: (El autor)

Arranque de Mininet

Arrancamos Mininet utilizando la aplicación que hemos creado, en nuestro caso *ruteo.py.*

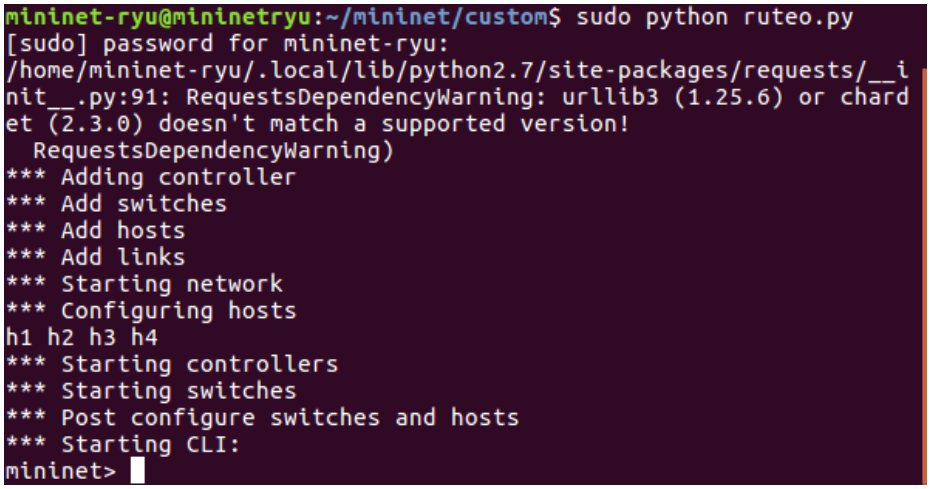


Figura 52. Comando Mininet para arrancar la topología creada.

Fuente: (El autor)

Se puede utilizar algunos comandos de Mininet como ***links*** para ver la conexión establecida entre todos los elementos de red, ***nodes*** para ver todos los elementos que forman parte de la red y el comando ***net*** para observar la forma en que se encuentran conectados los elementos que forman parte de la topología (ver figura 53).

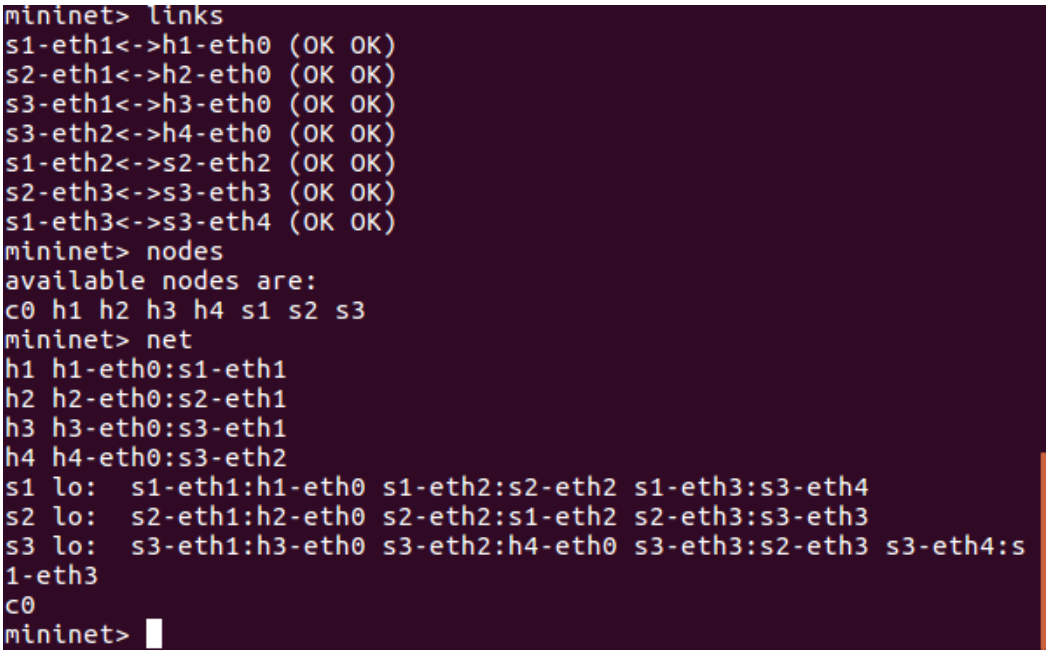


Figura 53. Comandos Mininet para observar el estado de la Topología lanzada.

Fuente: (El autor)

- **Configuración Comandos REST**

Se muestran las reglas de configuración REST utilizadas para Router:

Comandos para R1

curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:8080/router/0000000000000001

curl -X POST -d '{"address": "172.16.30.30/24"}' http://localhost:8080/router/0000000000000001

curl -X POST -d '{"address": "192.168.100.1/24"}' http://localhost:8080/router/0000000000000001

Comandos para R2

curl -X POST -d '{"address":"172.16.10.1/24"}' http://localhost:8080/router/0000000000000002

curl -X POST -d '{"address": "172.16.30.1/24"}' http://localhost:8080/router/0000000000000002

curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/router/0000000000000002

Comandos para R3

curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router/0000000000000003

curl -X POST -d '{"address": "192.168.10.20/24"}' http://localhost:8080/router/0000000000000003

curl -X POST -d '{"address": "192.168.100.20/24"}' http://localhost:8080/router/0000000000000003

Configuración de Gateways

curl -X POST -d '{"gateway": "172.16.30.1"}' http://localhost:8080/router/0000000000000001

curl -X POST -d '{"gateway": "172.16.30.30"}' http://localhost:8080/router/0000000000000002

curl -X POST -d '{"gateway": "192.168.10.1"}' http://localhost:8080/router/0000000000000003

Resultados:

- **Verificación de la red creada utilizando la herramienta Wireshark**

  Prueba de Eco entre h1 y h2

  Las figuras 54 y 55 muestran el ping entre h1 y h2 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h1 ping h2
PING 172.16.10.10 (172.16.10.10) 56(84) bytes of data.
64 bytes from 172.16.10.10: icmp_seq=1 ttl=62 time=10.8 ms
64 bytes from 172.16.10.10: icmp_seq=2 ttl=62 time=0.391 ms
64 bytes from 172.16.10.10: icmp_seq=3 ttl=62 time=0.100 ms
64 bytes from 172.16.10.10: icmp_seq=4 ttl=62 time=0.101 ms
64 bytes from 172.16.10.10: icmp_seq=5 ttl=62 time=0.083 ms
64 bytes from 172.16.10.10: icmp_seq=6 ttl=62 time=0.096 ms
64 bytes from 172.16.10.10: icmp_seq=7 ttl=62 time=0.100 ms
64 bytes from 172.16.10.10: icmp_seq=8 ttl=62 time=0.098 ms
64 bytes from 172.16.10.10: icmp_seq=9 ttl=62 time=0.107 ms
```

Figura 54. Ping entre h1 y h2.

Fuente: (El autor)

90	182.795151008	172.16.20.10	172.16.10.10	ICMP	98 Echo
91	182.795187824	172.16.10.10	172.16.20.10	ICMP	98 Echo
92	183.819180634	172.16.20.10	172.16.10.10	ICMP	98 Echo
93	183.819221148	172.16.10.10	172.16.20.10	ICMP	98 Echo
94	184.843203562	172.16.20.10	172.16.10.10	ICMP	98 Echo
95	184.843239015	172.16.10.10	172.16.20.10	ICMP	98 Echo
96	185.867108065	172.16.20.10	172.16.10.10	ICMP	98 Echo
97	185.867129939	172.16.10.10	172.16.20.10	ICMP	98 Echo
98	186.891166706	172.16.20.10	172.16.10.10	ICMP	98 Echo
99	186.891203275	172.16.10.10	172.16.20.10	ICMP	98 Echo
100	187.915207841	172.16.20.10	172.16.10.10	ICMP	98 Echo
101	187.915244279	172.16.10.10	172.16.20.10	ICMP	98 Echo
102	188.939199549	172.16.20.10	172.16.10.10	ICMP	98 Echo
103	188.939234451	172.16.10.10	172.16.20.10	ICMP	98 Echo
104	189.963207109	172.16.20.10	172.16.10.10	ICMP	98 Echo
105	189.963242281	172.16.10.10	172.16.20.10	ICMP	98 Echo
106	190.987208863	172.16.20.10	172.16.10.10	ICMP	98 Echo
107	190.987244309	172.16.10.10	172.16.20.10	ICMP	98 Echo

Figura 55. Capturas de tráfico en Wireshark entre h1 y h2.

Fuente: (El autor)

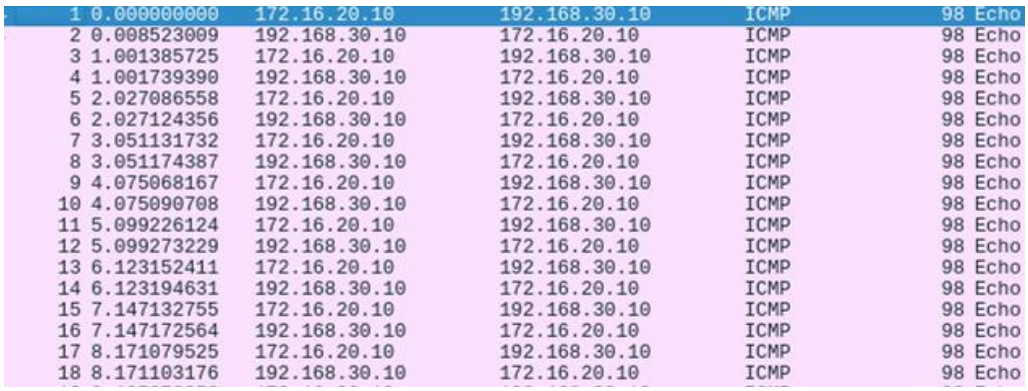
Prueba de Eco entre h1 y h3

Las figuras 56 y 57 muestran el ping entre h1 y h3 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h1 ping h3
PING 192.168.30.10 (192.168.30.10) 56(84) bytes of data.
64 bytes from 192.168.30.10: icmp_seq=1 ttl=61 time=17.5 ms
64 bytes from 192.168.30.10: icmp_seq=2 ttl=61 time=0.587 ms
64 bytes from 192.168.30.10: icmp_seq=3 ttl=61 time=0.073 ms
64 bytes from 192.168.30.10: icmp_seq=4 ttl=61 time=0.101 ms
64 bytes from 192.168.30.10: icmp_seq=5 ttl=61 time=0.052 ms
64 bytes from 192.168.30.10: icmp_seq=6 ttl=61 time=0.113 ms
64 bytes from 192.168.30.10: icmp_seq=7 ttl=61 time=0.099 ms
64 bytes from 192.168.30.10: icmp_seq=8 ttl=61 time=0.092 ms
64 bytes from 192.168.30.10: icmp_seq=9 ttl=61 time=0.055 ms
```

Figura 56. Ping entre h1 y h3.

Fuente: (El autor)



1	0.000000000	172.16.20.10	192.168.30.10	ICMP	98 Echo
2	0.008523009	192.168.30.10	172.16.20.10	ICMP	98 Echo
3	1.001385725	172.16.20.10	192.168.30.10	ICMP	98 Echo
4	1.001739390	192.168.30.10	172.16.20.10	ICMP	98 Echo
5	2.027086558	172.16.20.10	192.168.30.10	ICMP	98 Echo
6	2.027124356	192.168.30.10	172.16.20.10	ICMP	98 Echo
7	3.051131732	172.16.20.10	192.168.30.10	ICMP	98 Echo
8	3.051174387	192.168.30.10	172.16.20.10	ICMP	98 Echo
9	4.075068167	172.16.20.10	192.168.30.10	ICMP	98 Echo
10	4.075090708	192.168.30.10	172.16.20.10	ICMP	98 Echo
11	5.099226124	172.16.20.10	192.168.30.10	ICMP	98 Echo
12	5.099273229	192.168.30.10	172.16.20.10	ICMP	98 Echo
13	6.123152411	172.16.20.10	192.168.30.10	ICMP	98 Echo
14	6.123194631	192.168.30.10	172.16.20.10	ICMP	98 Echo
15	7.147132755	172.16.20.10	192.168.30.10	ICMP	98 Echo
16	7.147172564	192.168.30.10	172.16.20.10	ICMP	98 Echo
17	8.171079525	172.16.20.10	192.168.30.10	ICMP	98 Echo
18	8.171103176	192.168.30.10	172.16.20.10	ICMP	98 Echo

Figura 57. Capturas de tráfico en Wireshark entre h1 y h3.

Fuente: (El autor)

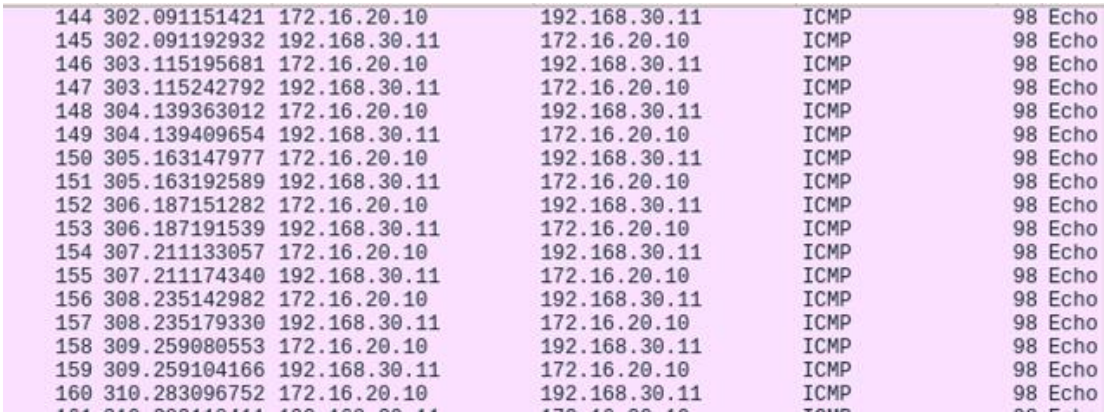
Prueba de Eco entre h1 y h4

Las figuras 58 y 59 muestran el ping entre h1 y h4 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h1 ping h4
PING 192.168.30.11 (192.168.30.11) 56(84) bytes of data.
64 bytes from 192.168.30.11: icmp_seq=1 ttl=61 time=9.83 ms
64 bytes from 192.168.30.11: icmp_seq=2 ttl=61 time=0.421 ms
64 bytes from 192.168.30.11: icmp_seq=3 ttl=61 time=0.120 ms
64 bytes from 192.168.30.11: icmp_seq=4 ttl=61 time=0.115 ms
64 bytes from 192.168.30.11: icmp_seq=5 ttl=61 time=0.115 ms
64 bytes from 192.168.30.11: icmp_seq=6 ttl=61 time=0.116 ms
64 bytes from 192.168.30.11: icmp_seq=7 ttl=61 time=0.113 ms
64 bytes from 192.168.30.11: icmp_seq=8 ttl=61 time=0.111 ms
64 bytes from 192.168.30.11: icmp_seq=9 ttl=61 time=0.116 ms
```

Figura 58. Ping entre h1 y h4.

Fuente: (El autor)



144	302.091151421	172.16.20.10	192.168.30.11	ICMP	98 Echo
145	302.091192932	192.168.30.11	172.16.20.10	ICMP	98 Echo
146	303.115195681	172.16.20.10	192.168.30.11	ICMP	98 Echo
147	303.115242792	192.168.30.11	172.16.20.10	ICMP	98 Echo
148	304.139363012	172.16.20.10	192.168.30.11	ICMP	98 Echo
149	304.139409654	192.168.30.11	172.16.20.10	ICMP	98 Echo
150	305.163147977	172.16.20.10	192.168.30.11	ICMP	98 Echo
151	305.163192589	192.168.30.11	172.16.20.10	ICMP	98 Echo
152	306.187151282	172.16.20.10	192.168.30.11	ICMP	98 Echo
153	306.187191539	192.168.30.11	172.16.20.10	ICMP	98 Echo
154	307.211133057	172.16.20.10	192.168.30.11	ICMP	98 Echo
155	307.211174340	192.168.30.11	172.16.20.10	ICMP	98 Echo
156	308.235142982	172.16.20.10	192.168.30.11	ICMP	98 Echo
157	308.235179330	192.168.30.11	172.16.20.10	ICMP	98 Echo
158	309.259080553	172.16.20.10	192.168.30.11	ICMP	98 Echo
159	309.259104166	192.168.30.11	172.16.20.10	ICMP	98 Echo
160	310.283096752	172.16.20.10	192.168.30.11	ICMP	98 Echo

Figura 59. Capturas de tráfico en Wireshark entre h1 y h4.

Fuente: (El autor)

Prueba de Eco entre h2 y h3

Las figuras 60 y 61 muestran el ping entre h2 y h3 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h2 ping h3
PING 192.168.30.10 (192.168.30.10) 56(84) bytes of data.
64 bytes from 192.168.30.10: icmp_seq=1 ttl=62 time=0.636 ms
64 bytes from 192.168.30.10: icmp_seq=2 ttl=62 time=0.087 ms
64 bytes from 192.168.30.10: icmp_seq=3 ttl=62 time=0.097 ms
64 bytes from 192.168.30.10: icmp_seq=4 ttl=62 time=0.098 ms
64 bytes from 192.168.30.10: icmp_seq=5 ttl=62 time=0.092 ms
64 bytes from 192.168.30.10: icmp_seq=6 ttl=62 time=0.104 ms
64 bytes from 192.168.30.10: icmp_seq=7 ttl=62 time=0.092 ms
64 bytes from 192.168.30.10: icmp_seq=8 ttl=62 time=0.092 ms
64 bytes from 192.168.30.10: icmp_seq=9 ttl=62 time=0.101 ms
```

Figura 60. Ping entre h2 y h3.

Fuente: (El autor)

15	7.168008746	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
16	7.168041730	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
17	8.192137192	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
18	8.192176119	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
19	9.216091610	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
20	9.216138267	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
21	10.240086379	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
22	10.240124444	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
23	11.263962517	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
24	11.263981524	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
25	12.288016883	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
26	12.288043156	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
27	13.311970093	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
28	13.311988048	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …
29	14.336012174	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) request…
30	14.336045648	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) reply …

Figura 61. Capturas de tráfico en Wireshark entre h2 y h3.

Fuente: (El autor)

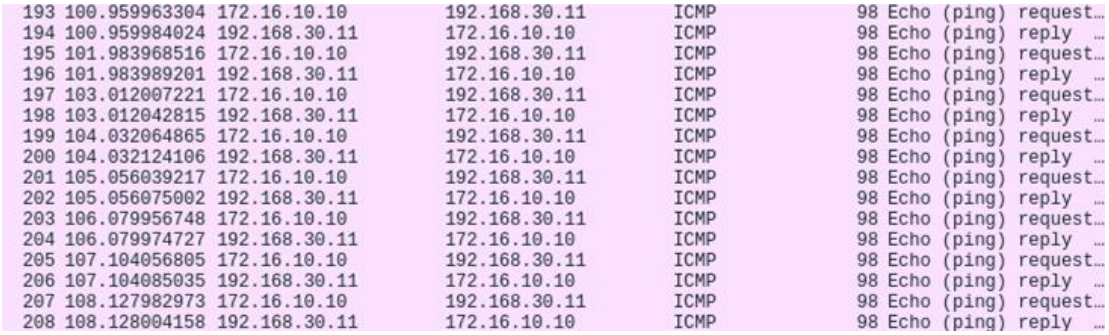
Prueba de Eco entre h2 y h4

Las figuras 62 y 63 muestran el ping entre h2 y h4 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h2 ping h4
PING 192.168.30.11 (192.168.30.11) 56(84) bytes of data.
64 bytes from 192.168.30.11: icmp_seq=1 ttl=62 time=0.242 ms
64 bytes from 192.168.30.11: icmp_seq=2 ttl=62 time=0.058 ms
64 bytes from 192.168.30.11: icmp_seq=3 ttl=62 time=0.102 ms
64 bytes from 192.168.30.11: icmp_seq=4 ttl=62 time=0.086 ms
64 bytes from 192.168.30.11: icmp_seq=5 ttl=62 time=0.050 ms
64 bytes from 192.168.30.11: icmp_seq=6 ttl=62 time=0.090 ms
64 bytes from 192.168.30.11: icmp_seq=7 ttl=62 time=0.067 ms
64 bytes from 192.168.30.11: icmp_seq=8 ttl=62 time=0.049 ms
64 bytes from 192.168.30.11: icmp_seq=9 ttl=62 time=0.047 ms
```

Figura 62. Ping entre h2 y h4.

Fuente: (El autor)



193	100.959963304	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
194	100.959984024	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
195	101.983968516	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
196	101.983989201	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
197	103.012007221	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
198	103.012042815	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
199	104.032064865	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
200	104.032124106	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
201	105.056039217	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
202	105.056075002	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
203	106.079956748	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
204	106.079974727	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
205	107.104056805	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
206	107.104085035	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …
207	108.127982973	172.16.10.10	192.168.30.11	ICMP	98 Echo (ping) request…
208	108.128004158	192.168.30.11	172.16.10.10	ICMP	98 Echo (ping) reply …

Figura 63. Capturas de tráfico en Wireshark entre h2 y h4.

Fuente: (El autor)

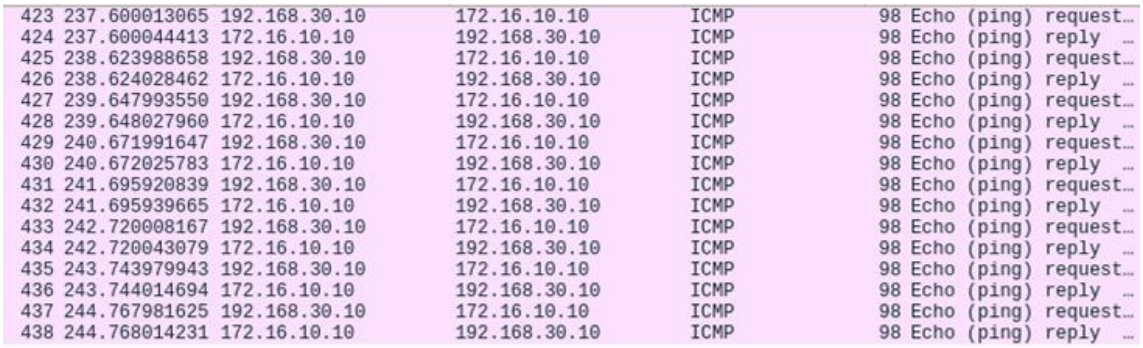
Prueba de Eco entre h3 y h2

Las figuras 64 y 65 muestran el ping entre h3 y h2 y la captura de tráfico con Wireshark respectivamente.

```
mininet> h3 ping h2
PING 172.16.10.10 (172.16.10.10) 56(84) bytes of data.
64 bytes from 172.16.10.10: icmp_seq=1 ttl=62 time=0.456 ms
64 bytes from 172.16.10.10: icmp_seq=2 ttl=62 time=0.101 ms
64 bytes from 172.16.10.10: icmp_seq=3 ttl=62 time=0.101 ms
64 bytes from 172.16.10.10: icmp_seq=4 ttl=62 time=0.100 ms
64 bytes from 172.16.10.10: icmp_seq=5 ttl=62 time=0.101 ms
64 bytes from 172.16.10.10: icmp_seq=6 ttl=62 time=0.088 ms
64 bytes from 172.16.10.10: icmp_seq=7 ttl=62 time=0.099 ms
64 bytes from 172.16.10.10: icmp_seq=8 ttl=62 time=0.095 ms
64 bytes from 172.16.10.10: icmp_seq=9 ttl=62 time=0.100 ms
```

Figura 64. Ping entre h3 y h2.

Fuente: (El autor)



423	237.600013065	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
424	237.600044413	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
425	238.623988658	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
426	238.624028462	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
427	239.647993550	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
428	239.648027960	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
429	240.671991647	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
430	240.672025783	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
431	241.695920839	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
432	241.695939665	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
433	242.720008167	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
434	242.720043079	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
435	243.743979943	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
436	243.744014694	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …
437	244.767981625	192.168.30.10	172.16.10.10	ICMP	98 Echo (ping) request…
438	244.768014231	172.16.10.10	192.168.30.10	ICMP	98 Echo (ping) reply …

Figura 65. Capturas de tráfico en Wireshark entre h3 y h2.

Fuente: (El autor)

Conclusiones:

- Se utilizó el simulador Mininet para crear r la topología de red, Ryu como controlador SDN, REST como protocolo de NorthBound y OpenFlow como protocolo de SouthBound.

- Se configuró un Switch dentro de la topología de red para que cumpla funciones de Router y arrancar la aplicación a través de Ryu, además de configurar las direcciones IP de las interfaces utilizadas.

Recomendaciones:

- Se recomienda realizar las pruebas de eco que se crean convenientes para determinar el correcto funcionamiento de la red.
- Es recomendable utilizar una topología base proporcionada por Mininet con la finalidad de poder utilizarla para el desarrollo de la práctica.

6.1.4 Práctica 4:

IPv6 en Mininet

Objetivos:

- Utilizar Mininet como herramienta de simulación para crear la topología utilizada en el desarrollo de esta práctica y crear una aplicación en Python que defina el funcionamiento de la red.
- Configurar direcciones IPv6 a las interfaces que conforman la topología de red y realizar pruebas con la herramienta Wireshark.

Recursos utilizados:

Tabla 13. Recursos Practica 4 Mininet

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Simulador Mininet - Controlador Ryu - OpenFlow v1.3 - Aplicación Python

Fuente: (El autor)

Procedimiento:

La topología utilizada para el desarrollo de esta práctica se muestra en la figura 66:

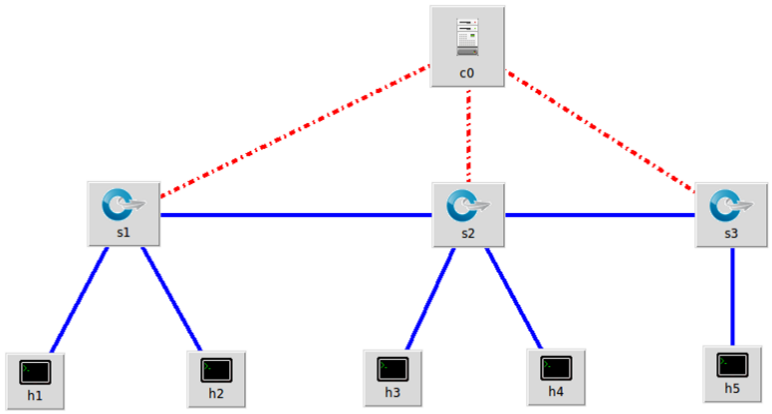


Figura 66. Topología de red utilizada para Practica IPv6.

Fuente: (El autor)

Se puede importar el código directamente desde Mininet a través de la opción Miniedit que es donde realmente se crea la topología, esto se observa en la figura 67.

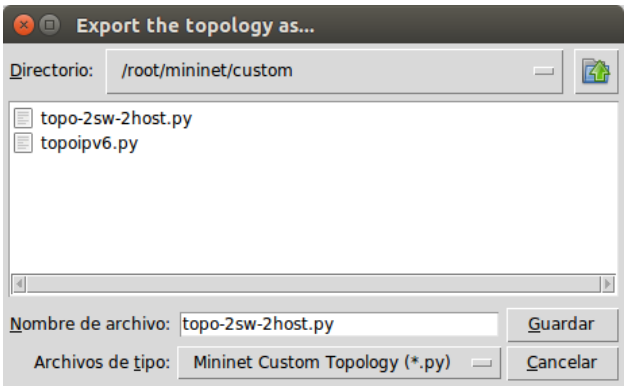


Figura 67. Almacenamiento de Topologías Miniedit

Fuente: (El autor)

El código utilizado para esta práctica se muestra en el Anexo 2 de este documento.

Es posible de observar el estado de la red utilizando los comandos ***net*** y ***links***. Este resultado se observa en la figura 68.

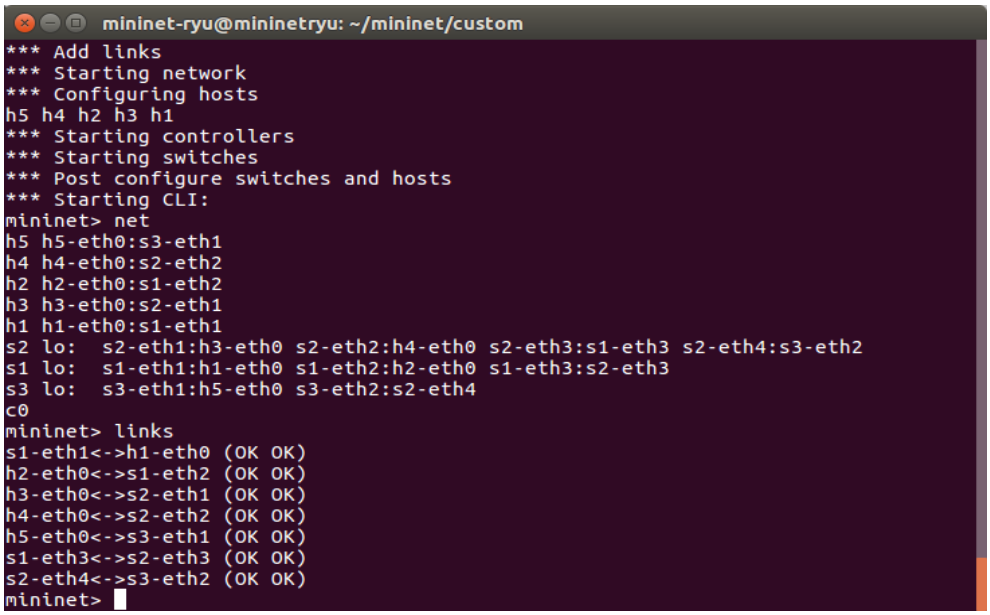


Figura 68. Comandos Mininet para ver el estado de la Red.

Fuente: (El autor)

- **Configuracion de Direccion IPv6**

  Una vez lanzada la topología que define la red para la práctica, se configura las direcciones IPv6 para cada elemento de la red, para ello se utiliza los comandos siguientes:

  >h1 ifconfig h1-eth0 inet6 add fc00::1/64

  >h2 ifconfig h2-eth0 inet6 add fc00::2/64

  >h3 ifconfig h3-eth0 inet6 add fc00::3/64

  >h4 ifconfig h4-eth0 inet6 add fc00::4/64

  >h5 ifconfig h5-eth0 inet6 add fc00::5/64

Resultados:

- **Pruebas de Eco con Wireshark**

  Las pruebas de funcionamiento se pueden hacer desde dos escenarios, el primero directamente desde la consola de Mininet y la segunda utilizando el terminal de cada host.

1. Pruebas en la consola de Mininet

> h1 ping6 fc00::2 -I h1-eth0

>h1 ping6 fc00::3 -I h1-eth0

>h1 ping6 fc00::5 -I h1-eth0

2. Realizando pruebas desde la consola de cada host

ping6 fc00::2 -I h1-eth0
ping6 fc00::3 -I h1-eth0
ping6 fc00::5 -I h1-eth0

Prueba de Eco entre h1 y h2

Las figuras 69 y 70 muestran el ping entre h1 y h2 y el análisis de tráfico en Wireshark respectivamente.

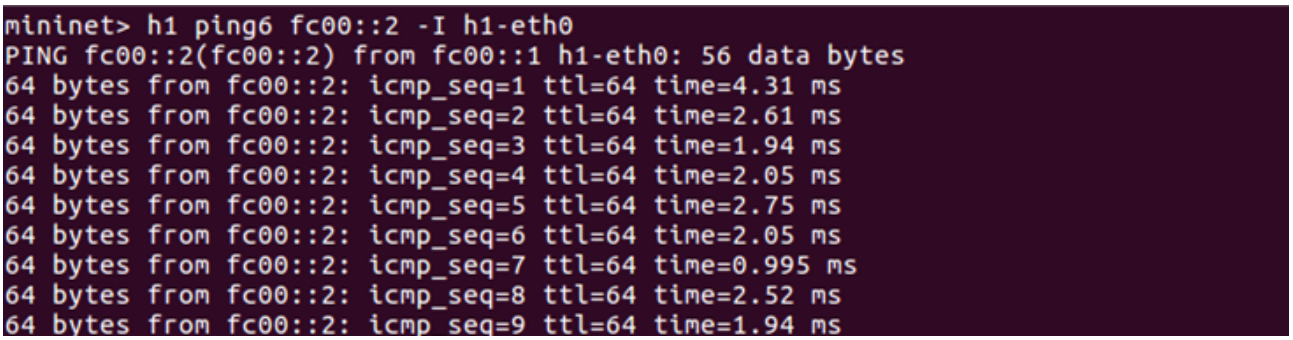


Figura 69. Ping entre h1 y h2 IPv6.

Fuente: (El autor)

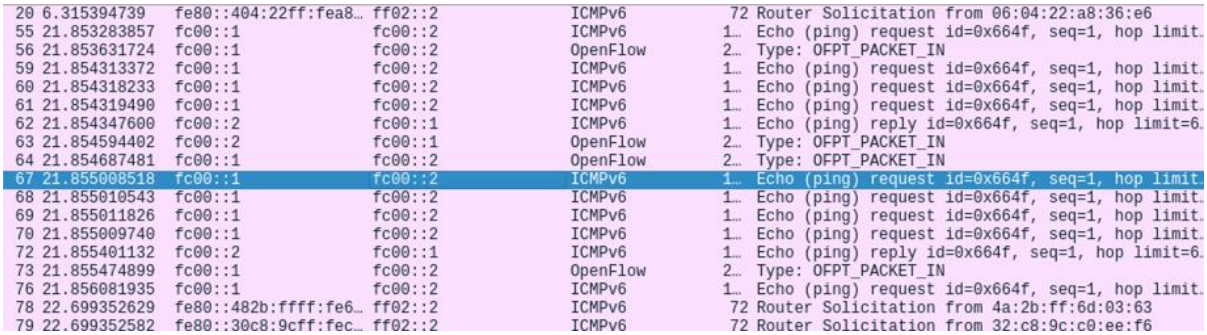


Figura 70. Capturas de tráfico en Wireshark entre h1 y h2 IPv6.

Fuente: (El autor)

Prueba de Eco entre h1 y h3

Las figuras 71 y 72 muestran el ping entre h1 y h3 y el análisis de tráfico en Wireshark respectivamente.

```
mininet> h1 ping6 fc00::3 -I h1-eth0
PING fc00::3(fc00::3) from fc00::1 h1-eth0: 56 data bytes
64 bytes from fc00::3: icmp_seq=1 ttl=64 time=6.94 ms
64 bytes from fc00::3: icmp_seq=2 ttl=64 time=4.34 ms
64 bytes from fc00::3: icmp_seq=3 ttl=64 time=1.91 ms
64 bytes from fc00::3: icmp_seq=4 ttl=64 time=1.78 ms
64 bytes from fc00::3: icmp_seq=5 ttl=64 time=3.56 ms
64 bytes from fc00::3: icmp_seq=6 ttl=64 time=3.84 ms
64 bytes from fc00::3: icmp_seq=7 ttl=64 time=3.60 ms
64 bytes from fc00::3: icmp_seq=8 ttl=64 time=3.12 ms
64 bytes from fc00::3: icmp_seq=9 ttl=64 time=4.57 ms
```

Figura 71. Ping entre h1 y h3.

Fuente: (El autor)

261	37.292746335	fe80::a8e3:6fff:fed…	fe80::601e:49ff:fe5…	ICMPv6	80	Neighbor Advertisement fe80::a8e3:6fff:fed4:d77…
262	37.293088711	fe80::a8e3:6fff:fed…	fe80::601e:49ff:fe5…	OpenFlow	1…	Type: OFPT_PACKET_IN
264	37.293423743	fe80::a8e3:6fff:fed…	fe80::601e:49ff:fe5…	ICMPv6	80	Neighbor Advertisement fe80::a8e3:6fff:fed4:d77…
266	38.760547659	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
267	38.760944317	fc00::1	fc00::3	OpenFlow	2…	Type: OFPT_PACKET_IN
270	38.761523623	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
271	38.761528752	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
272	38.761529820	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
273	38.761859598	fc00::1	fc00::3	OpenFlow	2…	Type: OFPT_PACKET_IN
276	38.762354648	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
277	38.762359986	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
278	38.762362241	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
279	38.762358326	fc00::1	fc00::3	ICMPv6	1…	Echo (ping) request id=0x6655, seq=1, hop limit…
280	38.762417083	fc00::3	fc00::1	ICMPv6	1…	Echo (ping) reply id=0x6655, seq=1, hop limit=6…
281	38.762716673	fc00::1	fc00::3	OpenFlow	2…	Type: OFPT_PACKET_IN
282	38.762797265	fc00::3	fc00::1	OpenFlow	2…	Type: OFPT_PACKET_IN
285	38.763209102	fc00::3	fc00::1	ICMPv6	1…	Echo (ping) reply id=0x6655, seq=1, hop limit=64
286	38.763211687	fc00::3	fc00::1	ICMPv6	1…	Echo (ping) reply id=0x6655, seq=1, hop limit=64

Figura 72. Capturas de tráfico en Wireshark entre h1 y h3 IPv6.

Fuente: (El autor)

Prueba de Eco entre h1 y h4

Las figuras 73 y 74 muestran el ping entre h1 y h4 y el análisis de tráfico en Wireshark respectivamente.

```
mininet> h1 ping6 fc00::4 -I h1-eth0
PING fc00::4(fc00::4) from fc00::1 h1-eth0: 56 data bytes
64 bytes from fc00::4: icmp_seq=1 ttl=64 time=7.63 ms
64 bytes from fc00::4: icmp_seq=2 ttl=64 time=5.32 ms
64 bytes from fc00::4: icmp_seq=3 ttl=64 time=4.25 ms
64 bytes from fc00::4: icmp_seq=4 ttl=64 time=25.7 ms
64 bytes from fc00::4: icmp_seq=5 ttl=64 time=4.69 ms
64 bytes from fc00::4: icmp_seq=6 ttl=64 time=6.34 ms
64 bytes from fc00::4: icmp_seq=7 ttl=64 time=3.94 ms
64 bytes from fc00::4: icmp_seq=8 ttl=64 time=4.43 ms
64 bytes from fc00::4: icmp_seq=9 ttl=64 time=2.13 ms
```

Figura 73. Ping entre h1 y h4.

Fuente: (El autor)

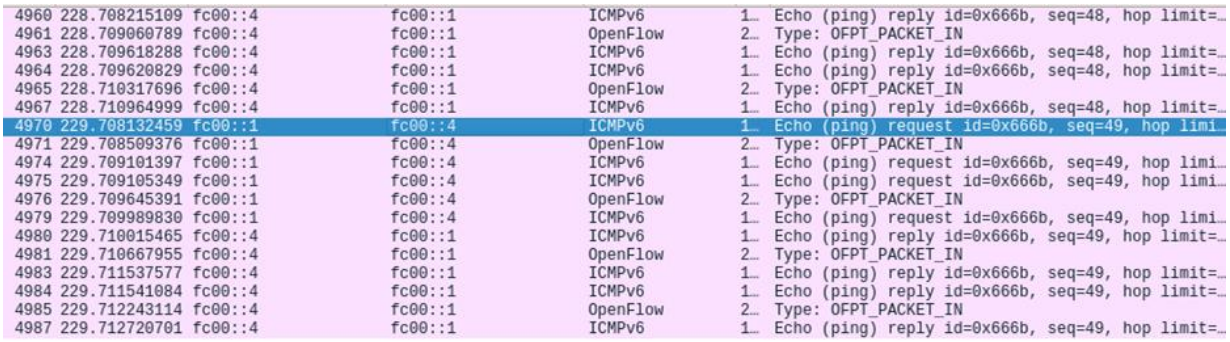


Figura 74. Capturas de tráfico en Wireshark entre h1 y h4 IPv6.

Fuente: (El autor)

- **Prueba de Eco entre h1 y h5**

  Las figuras 75 y 76 muestran el ping entre h1 y h5 y el análisis de tráfico en Wireshark respectivamente.

```
mininet> h1 ping6 fc00::5 -I h1-eth0
PING fc00::5(fc00::5) from fc00::1 h1-eth0: 56 data bytes
64 bytes from fc00::5: icmp_seq=1 ttl=64 time=9.53 ms
64 bytes from fc00::5: icmp_seq=2 ttl=64 time=4.78 ms
64 bytes from fc00::5: icmp_seq=3 ttl=64 time=3.41 ms
64 bytes from fc00::5: icmp_seq=4 ttl=64 time=3.04 ms
64 bytes from fc00::5: icmp_seq=5 ttl=64 time=6.22 ms
64 bytes from fc00::5: icmp_seq=6 ttl=64 time=5.12 ms
64 bytes from fc00::5: icmp_seq=7 ttl=64 time=5.14 ms
64 bytes from fc00::5: icmp_seq=8 ttl=64 time=5.04 ms
64 bytes from fc00::5: icmp_seq=9 ttl=64 time=5.33 ms
```

Figura 75. Ping ente h1 y h5.

Fuente: (El autor)

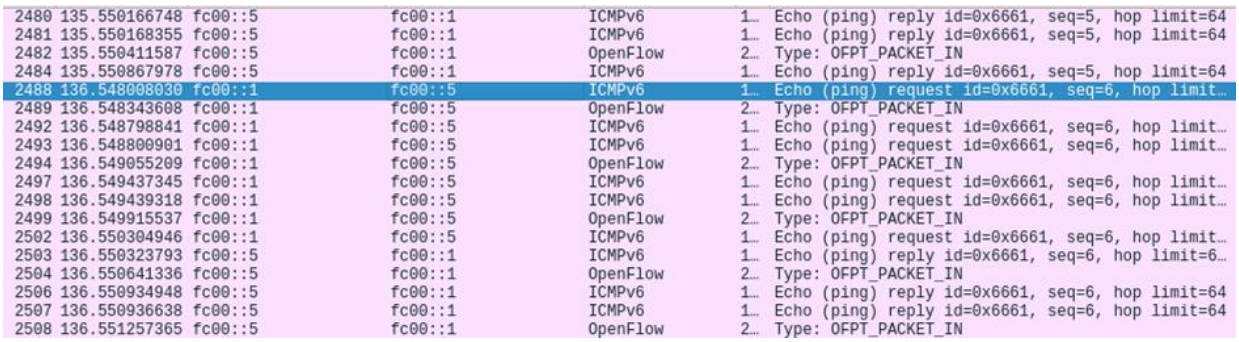


Figura 76. Capturas de tráfico en Wireshark entre h1 y h5 IPv6.

Fuente: (El autor)

Se puede apreciar también dentro de la consola de los Switch un resumen de la forma como se encuentran conectados los elementos en la red, esto se aprecia en la figura 77.

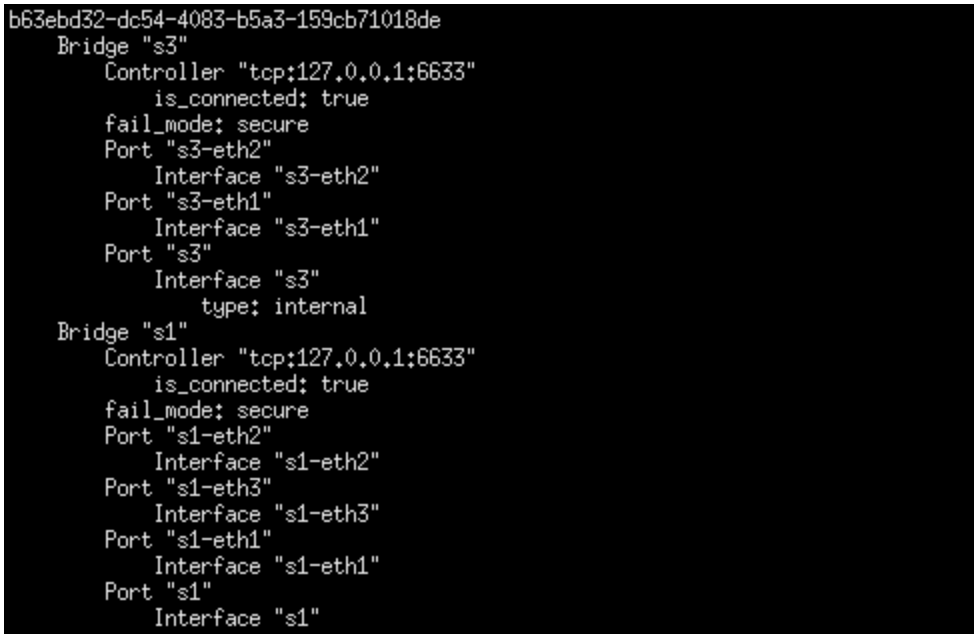


Figura 77. Interfaces de Salida de cada Switch.

Fuente: (El autor)

Conclusiones:

- Se utilizó Mininet como herramienta de simulación para crear la topología utilizada en el desarrollo de esta práctica y se creó, además una aplicación en Python que define el funcionamiento de la red.
- Se configuro direcciones IPv6 a las interfaces que conforman la topología de red y se realizó pruebas con la herramienta Wireshark para verificar el correcto funcionamiento de la misma.

Recomendaciones:

- Se recomienda mantener un orden durante la configuración de direcciones IP, de tal forma que se facilite el análisis de la red durante la ejecución de pruebas.
- Se recomienda utilizar una aplicación base proporcionada por Mininet para facilitar el desarrollo de la práctica.

6.1.5 PRACTICA 5:

TRANSMISIÓN DE VIDEO

Objetivos:

- Utilizar Mininet como herramienta de simulación para establecer la topología de red utilizada para la Práctica, Ryu como controlador SDN y Python como lenguaje de programación para crear la aplicación que defina el funcionamiento de la red.
- Utilizar el Reproductor Multimedia VLC para transmisión de video entre hosts dentro de la red.

Recursos utilizados:

Tabla 14. Recursos Practica 5 Mininet

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Simulador Mininet - Controlador Ryu - OpenFlow v1.3 - Aplicación Python - Reproductor Multimedia VLC

Fuente: (El autor)

Procedimiento:

La figura 78 muestra la topología utilizada para el desarrollo de esta Práctica.

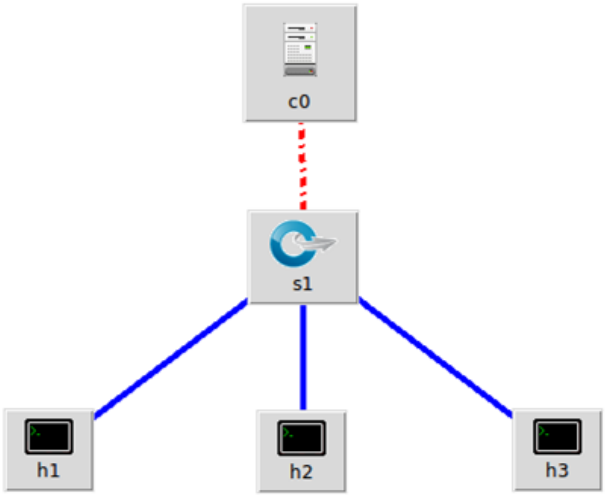


Figura 78. Topología de Red para Práctica de transmisión de Video.

Fuente: (El autor)

La aplicación utilizada para esta práctica se aprecia en el **Anexo 3** de este documento.

- **Arranque de Mininet**

Una vez creada la aplicación para la Práctica en cuestión que en nuestro caso se denomina *ejemplovideo.py* procedemos a correrla, esto se muestra en la figura 79.

```
mininet-ryu@mininetryu:~/mininet/custom$ sudo python ejemplovideo.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h3 h1 h2
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h3 -> h1 h2
h1 -> h3 h2
h2 -> h3 h1
*** Results: 0% dropped (6/6 received)
mininet> xterm h1 h2 h3
mininet>
```

Figura 79. Lanzamiento de la Aplicación para la Práctica de Video.

Fuente: (El autor)

- **Pruebas entre h1 y h2**

Se puede abrir una consola para cada host y empezar con las pruebas, esto a través del comando ***xterm*** como se muestra en la figura 80.

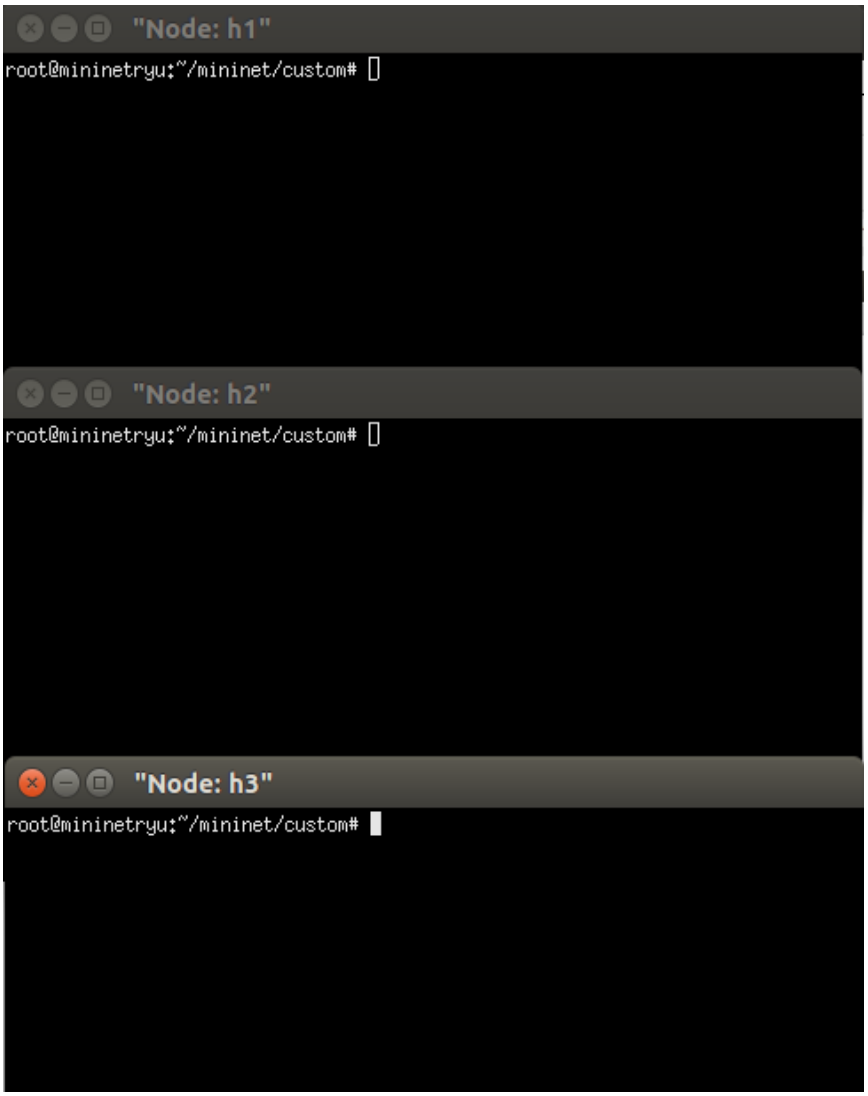


Figura 80. Apertura de Consola para cada host de la Red.

Fuente: (El autor)

- **Emisor H1**

Se debe seguir los siguientes pasos para realizar la transmisión de video.

Paso 1. En la consola de h1 que en este caso será el emisor de video, invocamos el reproductor VLC a través del comando *vlc-wrapper &.* Como resultado se nos abrirá el

reproductor de video donde empezaremos a realizar la configuración previa del host emisor. Esto se evidencia en la figura 81.

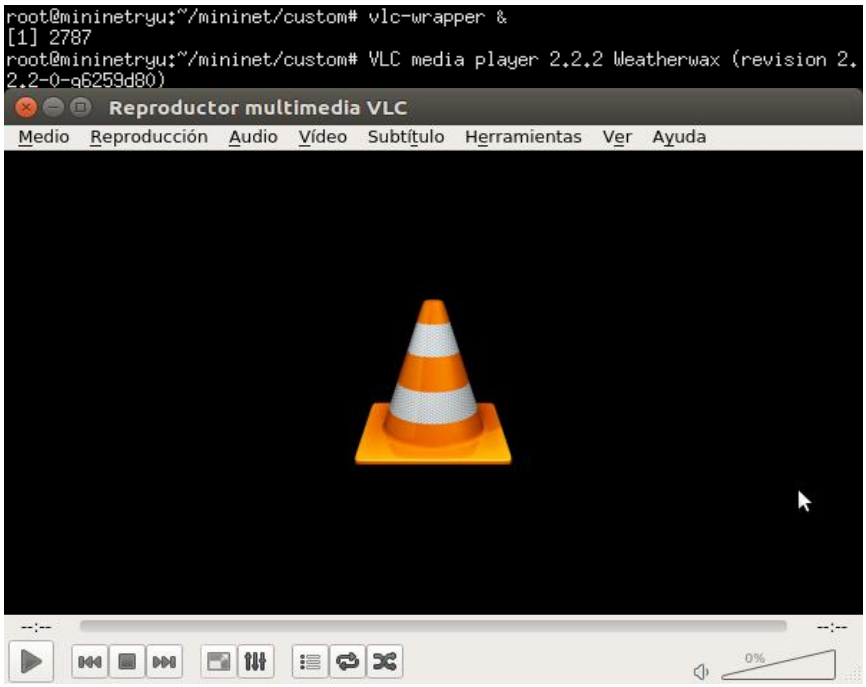


Figura 81. Apertura del Reproductor de Video VLC h1.

Fuente: (El autor)

Paso 2. Abrimos la opción medio del reproductor donde empezaremos a configurar algunos parámetros, esto se observa en la figura 82.

Figura 82. Parámetros de configuración del Reproductor VLC.

Fuente: (El autor)

Paso 3. En la opción añadir buscamos la ruta donde se encuentra el video que queremos transmitir al host receptor (ver figura 83).

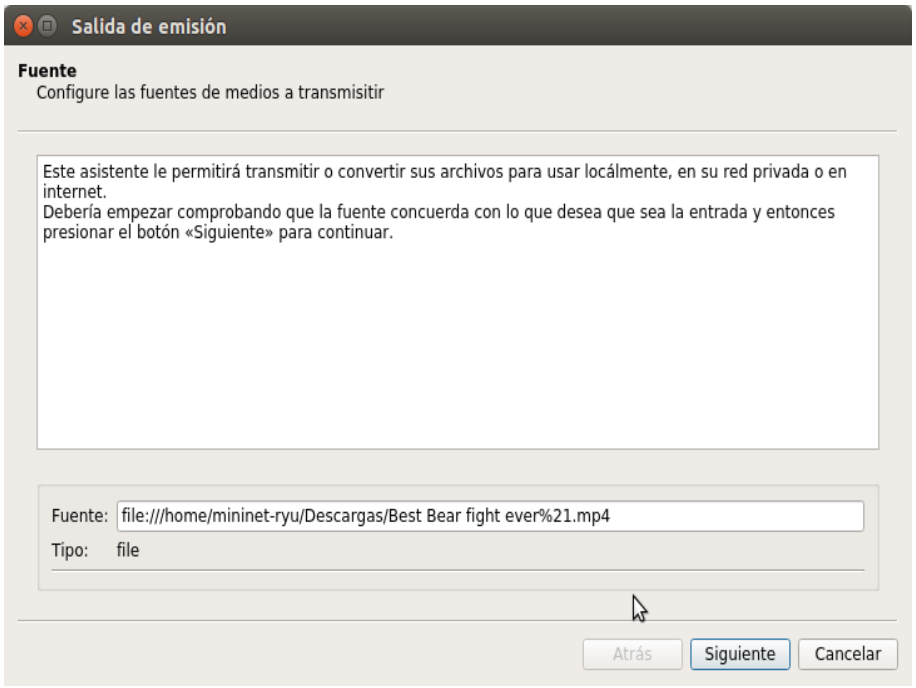


Figura 83. Ruta de video que se va a transmitir al host receptor.

Fuente: (El autor)

Paso 4. Es necesario también configurar el Protocolo de Transmisión en este caso RTP, como se muestra en la figura 84.

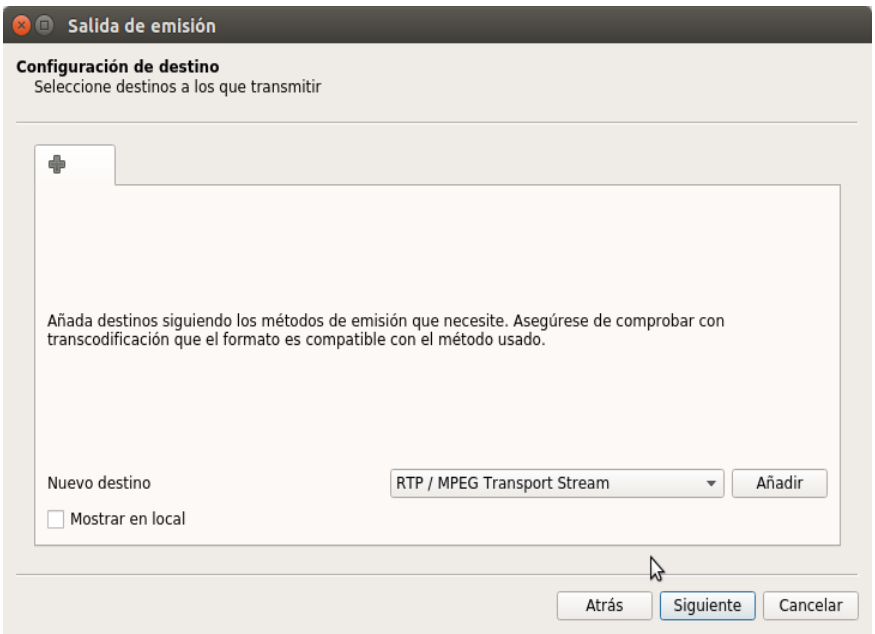


Figura 84. Selección de Protocolo de Transmisión.

Fuente: (El autor)

Paso 5. Se debe configurar la salida de emisión con la dirección IP del host al que se va a transmitir, un puerto base que debe coincidir con el equipo receptor y un nombre común que debemos darle a la transmisión.

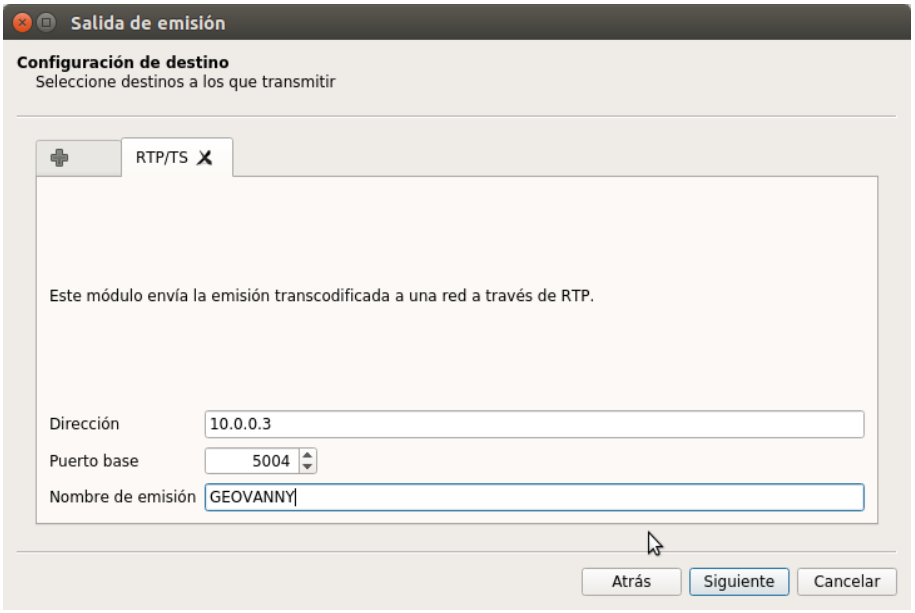


Figura 85. Configuración de IP, Puerto y Nombre de Transmisión h3.

Fuente: (El autor)

Paso 6. Por último, es necesario escoger el estándar de Video, en este caso SD (ver figura 86).

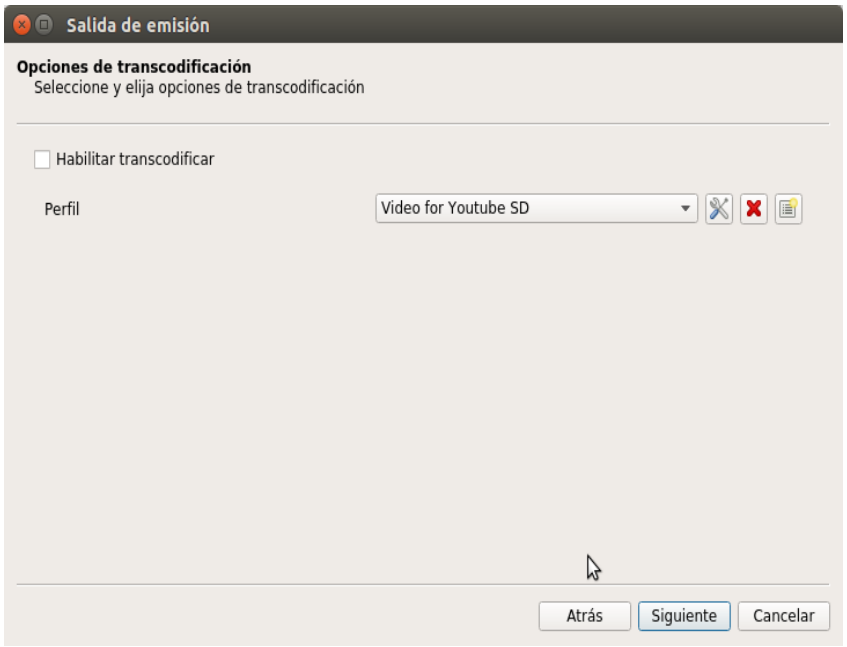


Figura 86. Estándar de Reproducción de Video.

Fuente: (El autor)

Paso 7. Tendremos listo nuestro host emisor para empezar a emitir el video seleccionado.

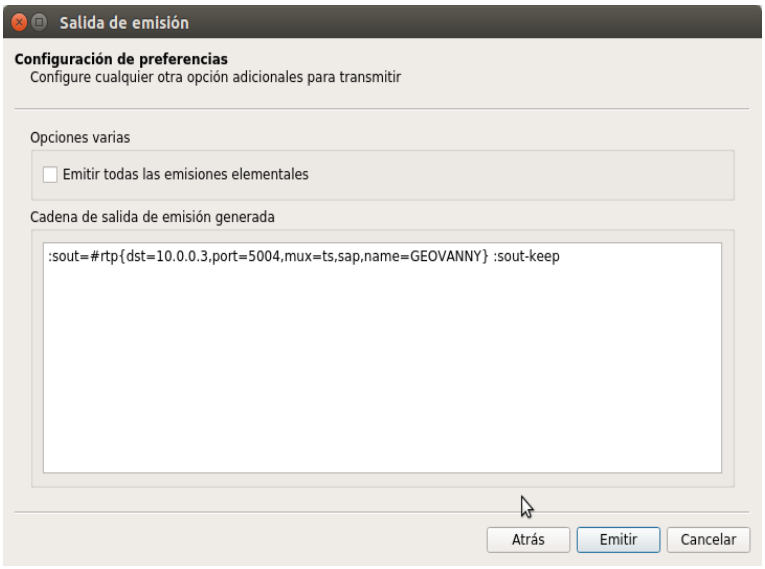


Figura 87. Emisor configurado y listo para transmitir al host3.

Fuente: (El autor)

- **Receptor H3**

Pasos a seguir para el host Receptor:

Paso 1. De igual forma desde la consola del host receptor, en este caso h3 invocamos al reproductor VLC como se muestra en la figura 88.

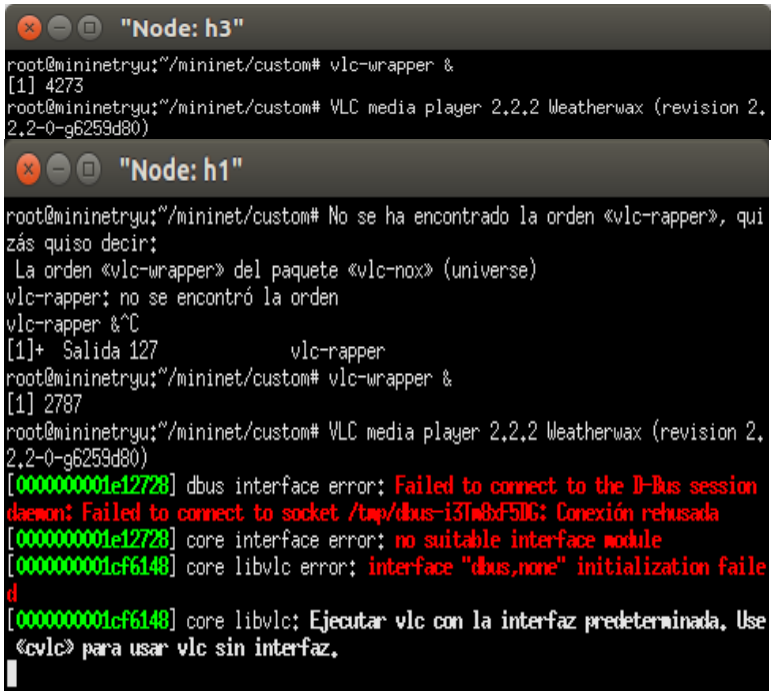


Figura 88. Apertura del reproductor de video VLC h3.

Fuente: (El autor)

Paso 2. En el equipo Receptor únicamente debemos tener presente el protocolo de Reproducción, la IP del host, el puerto base y el nombre que le asignamos a la transmisión. Todos estos parámetros se configuran en la opción Red (ver figura 89).

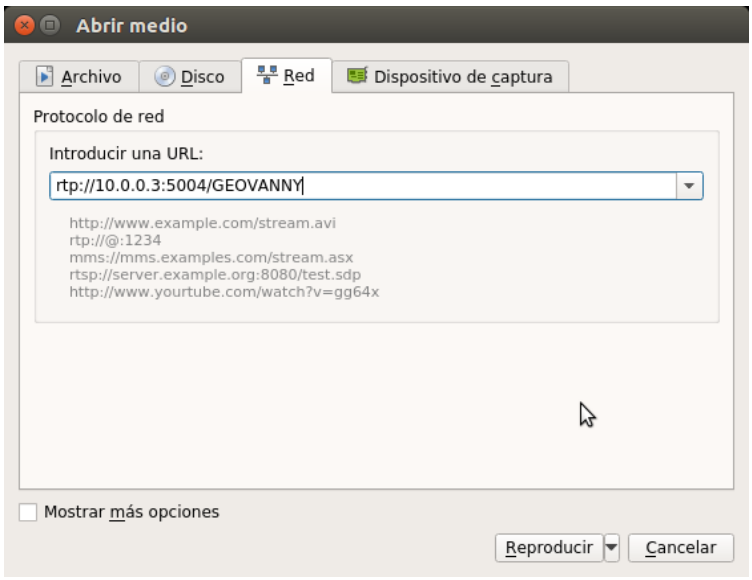


Figura 89. Configuración de equipo receptor h3.

Fuente: (El autor)

- **Receptor H2**

  Paso 1. Se puede seguir el mismo proceso para transmitir video hacia el host 2, en este caso lo único que se debe cambiar es la IP de destino como se muestra en la figura 90.

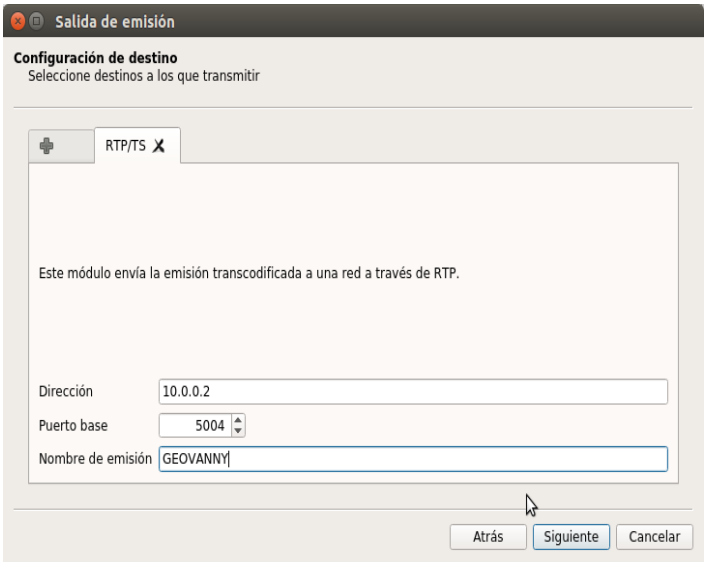


Figura 90. Configuración de IP, Puerto y Nombre de Transmisión.

Fuente: (El autor)

- **Paso 2.** Como resultado de esta configuración la salida de emisión nos quedara de la siguiente forma (ver figura 91).

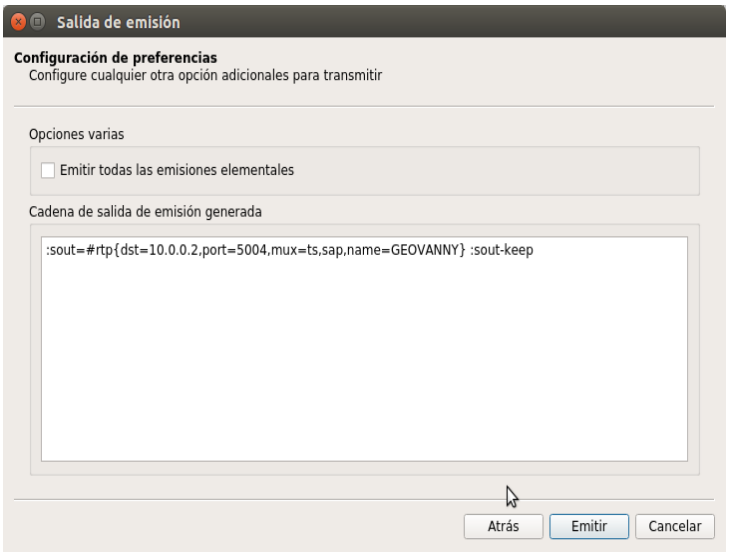


Figura 91. Emisor configurado y listo para transmitir al host 2.

Fuente: (El autor)

Resultados:

- **Resultados entre H1 y H3**

Los resultados de transmisión de video entre el Host 1 y el Host 3 utilizando el reproductor VLC se muestran en la figura 92, además del análisis de paquetes en Wireshark que se muestran en la figura 93.

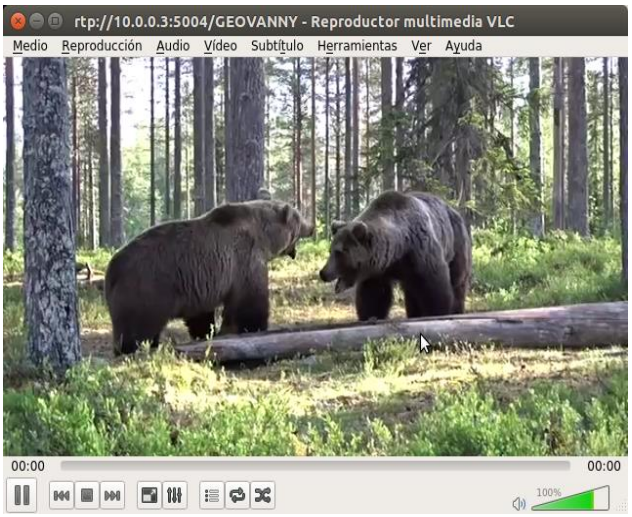
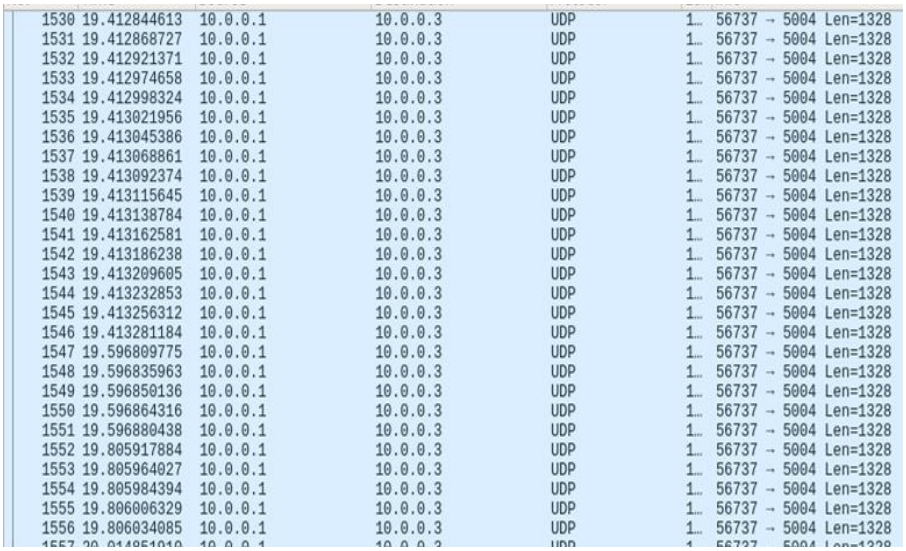


Figura 92. Reproducción de Video h3.

Fuente: (El autor)



1530	19.412844613	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1531	19.412868727	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1532	19.412921371	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1533	19.412974658	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1534	19.412998324	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1535	19.413021956	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1536	19.413045386	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1537	19.413068861	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1538	19.413092374	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1539	19.413115645	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1540	19.413138784	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1541	19.413162581	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1542	19.413186238	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1543	19.413209605	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1544	19.413232853	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1545	19.413256312	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1546	19.413281184	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1547	19.596809775	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1548	19.596835963	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1549	19.596850136	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1550	19.596864316	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1551	19.596880438	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1552	19.805917884	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1553	19.805964027	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1554	19.805984394	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1555	19.806006329	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328
1556	19.806034085	10.0.0.1	10.0.0.3	UDP	1… 56737 → 5004 Len=1328

Figura 93. Análisis de Transmisión de paquetes con Wireshark h3.

Fuente: (El autor)

- **Resultados entre H1 y H2**

Los resultados de reproducción de video para el Host 2 utilizando el reproductor VLC se muestran en la figura 94, además del análisis en Wireshark mostrado en la figura 95.

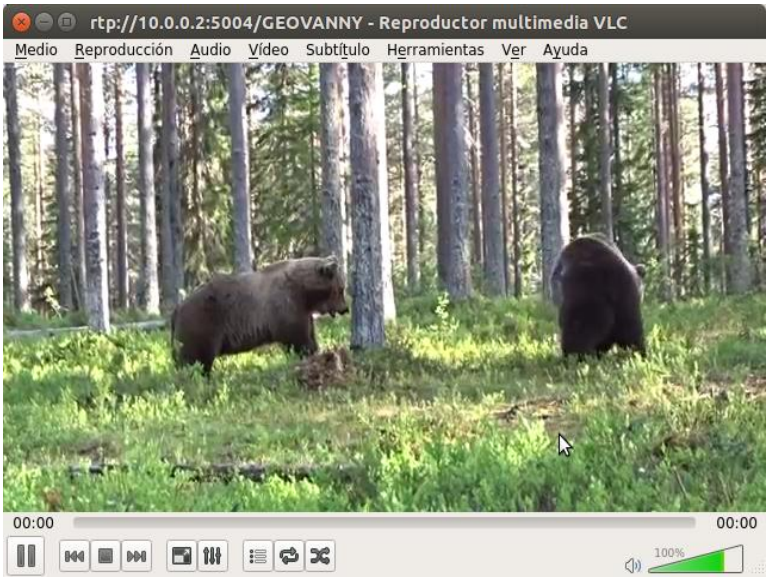
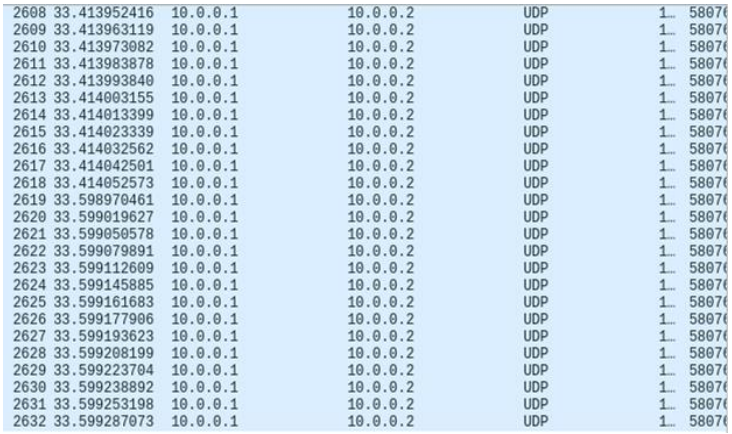


Figura 94. Reproducción de Video h2.

Fuente: (El autor)



2608	33.413952416	10.0.0.1	10.0.0.2	UDP	1…	5807
2609	33.413963119	10.0.0.1	10.0.0.2	UDP	1…	5807
2610	33.413973082	10.0.0.1	10.0.0.2	UDP	1…	5807
2611	33.413983878	10.0.0.1	10.0.0.2	UDP	1…	5807
2612	33.413993840	10.0.0.1	10.0.0.2	UDP	1…	5807
2613	33.414003155	10.0.0.1	10.0.0.2	UDP	1…	5807
2614	33.414013399	10.0.0.1	10.0.0.2	UDP	1…	5807
2615	33.414023339	10.0.0.1	10.0.0.2	UDP	1…	5807
2616	33.414032562	10.0.0.1	10.0.0.2	UDP	1…	5807
2617	33.414042501	10.0.0.1	10.0.0.2	UDP	1…	5807
2618	33.414052573	10.0.0.1	10.0.0.2	UDP	1…	5807
2619	33.598970461	10.0.0.1	10.0.0.2	UDP	1…	5807
2620	33.599019627	10.0.0.1	10.0.0.2	UDP	1…	5807
2621	33.599050578	10.0.0.1	10.0.0.2	UDP	1…	5807
2622	33.599079891	10.0.0.1	10.0.0.2	UDP	1…	5807
2623	33.599112609	10.0.0.1	10.0.0.2	UDP	1…	5807
2624	33.599145885	10.0.0.1	10.0.0.2	UDP	1…	5807
2625	33.599161683	10.0.0.1	10.0.0.2	UDP	1…	5807
2626	33.599177906	10.0.0.1	10.0.0.2	UDP	1…	5807
2627	33.599193623	10.0.0.1	10.0.0.2	UDP	1…	5807
2628	33.599208199	10.0.0.1	10.0.0.2	UDP	1…	5807
2629	33.599223704	10.0.0.1	10.0.0.2	UDP	1…	5807
2630	33.599238892	10.0.0.1	10.0.0.2	UDP	1…	5807
2631	33.599253198	10.0.0.1	10.0.0.2	UDP	1…	5807
2632	33.599287073	10.0.0.1	10.0.0.2	UDP	1…	5807

Figura 95. Análisis de Transmisión de paquetes con Wireshark h2.

Fuente: (El autor)

Es importante aclarar que cualquier host puede actuar como emisor o como receptor, lo único que se debe considerar es configurar los parámetros de IP, Puerto Base, Protocolo y nombre de Transmisión.

Conclusiones:

- Se utilizó Mininet como herramienta de simulación para establecer la topología de red, Ryu como controlador SDN y Python como lenguaje de programación para crear la aplicación que define el funcionamiento de la red.
- Se utilizó el Reproductor Multimedia VLC para transmisión de video entre hosts dentro de la red.

Recomendaciones:

- Se recomienda tener presente los parámetros de configuración como IP, Puerto Base, Protocolo de Transmisión y nombre asignado a la transmisión antes de realizar la transmisión de video a fin de evitar posibles errores durante la reproducción**.**
- Se recomienda utilizar topologías de red base proporcionadas por Mininet, así como la herramienta de captura de tráfico Wireshark para observar el comportamiento de los hosts durante la transmisión.

6.2 PRUEBAS CON SWITCH ZODIAC FX

En esta sección se plantea el desarrollo de cinco practicas utilizando el Switch Zodiac FX para dicho fin. De igual forma que en la sección anterior se utiliza Ryu como controlador SDN y APIs en lenguaje Python que definen cada una de las funciones que cumplirá el Switch antes mencionado. De igual forma se hace uso del esquema utilizado con el Simulador Mininet para el desarrollo de las prácticas propuestas.

6.2.1 PRACTICA 1:

CONFIGURACION DE SWITCH ZODIAC FX

Objetivos:

- Actualizar al firmware del dispositivo Zodiac FX a su versión más reciente.
- Conocer y funcionamiento del Switch Zodiac FX, así como de sus componentes**.**

Recursos utilizados:

Tabla 15. Recursos Practica 1 Zodiac FX

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7.	- Sistema Operativo Ubuntu 16.04 - Archivo de actualización de firmware - OpenFlow v1.3 - SAM-BA - Putty

Fuente: (El autor)

Procedimiento:

- **Actualización de firmware**

Existen dos métodos para actualización de firmware del Zodiac FX, el primero vía Interfaz web y el segundo mediante el uso de SAM-BA que es un software que nos ayuda a cargar el firmware del Zodiac en su versión más reciente.

El método que utilizaremos es a través del programa SAM-BA para lo cual se realiza los siguientes pasos.

1. Descargar el archivo ***.bin*** de actualización de firmware disponible en el repositorio oficial de Zodiac FX https://northboundnetworks.com/.
2. Asumiendo que el dispositivo Zodiac FX se encuentra conectado mediante el puerto USB a nuestra PC, procedemos a desconectarlo.
3. Mientras el dispositivo esta apagado cerramos el puente del Zodiac FX en el indicador ERASE Jumper, como se muestra en la figura 96.

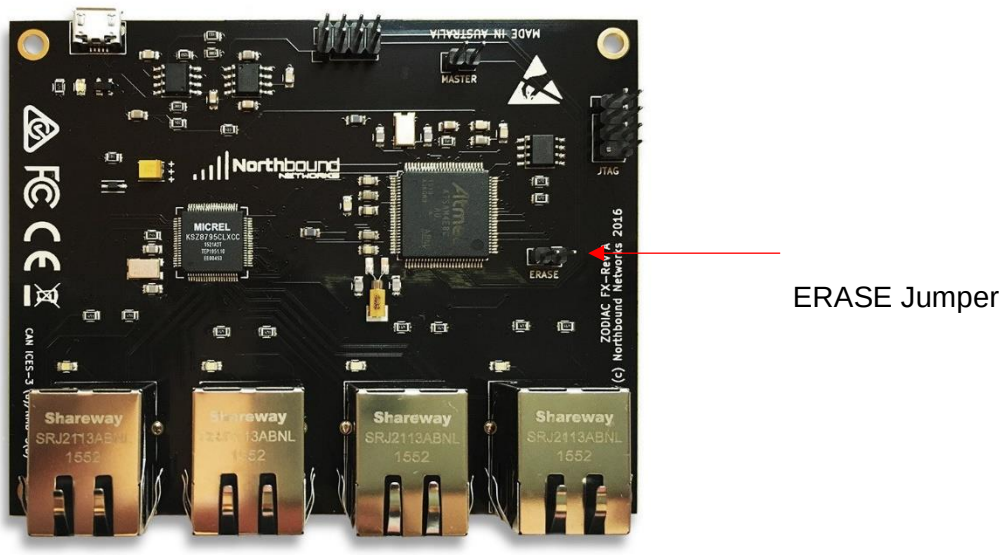


Figura 96. Cerrar Puente Zodiac FX.

Fuente: (El autor)

4. Volvemos a conectar la fuente de alimentación USB, esperamos un tiempo promedio de 5 a 10 segundos y la desconectamos nuevamente, este proceso borrara el firmware por defecto.
5. Movemos el puente a la posición abierta, es decir a la posición original.

6. Conectamos el cable USB nuevamente para encender el dispositivo y abrimos el programa SAM-BA, para ello debemos asegurarnos que el puerto COM sea correcto (consultar administración de dispositivos - Puertos COM en nuestra PC), y que el tipo de placa sea “at91sam4e8-ek” ver figuras 97 y 98.

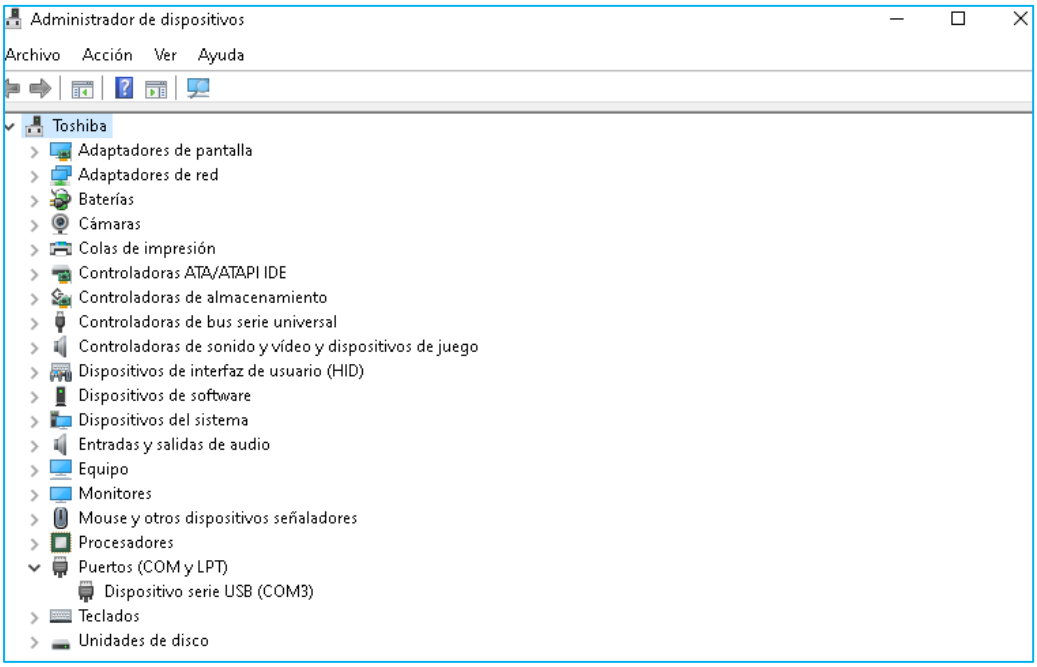


Figura 97. Puerto COM.

Fuente: (El autor)

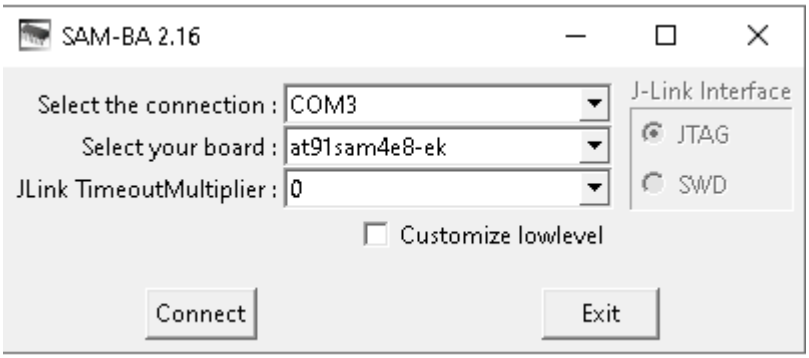


Figura 98. Selección de puerto COM y tipo de placa.

Fuente: (El autor)

7. Una vez conectados a nuestro dispositivo se nos abrirá una interfaz donde debemos cargar el archivo ***.bin*** del firmware que hemos descargado, y pulsamos el botón *send file*. Esto se muestra en la figura 99.

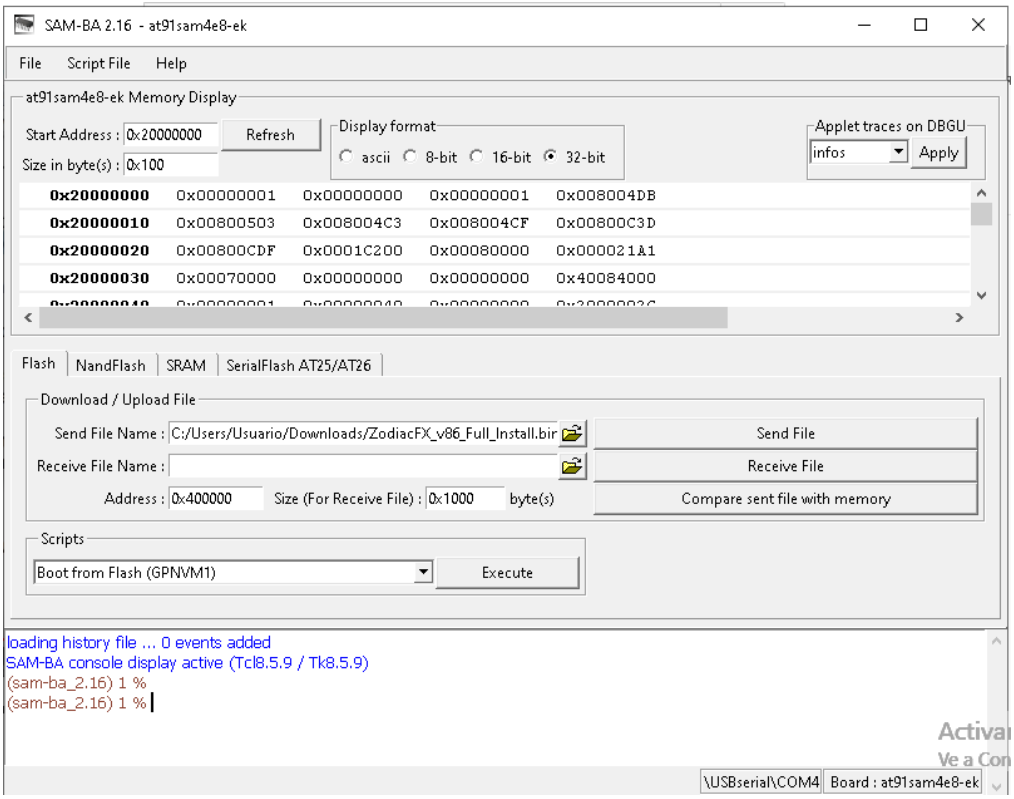


Figura 99. Archivo .bin del firmware.

Fuente: (El autor)

8. Cuando aparezca una ventana emergente que nos pida bloquear región, pulsamos la opción **NO** como se aprecia en la figura 100.

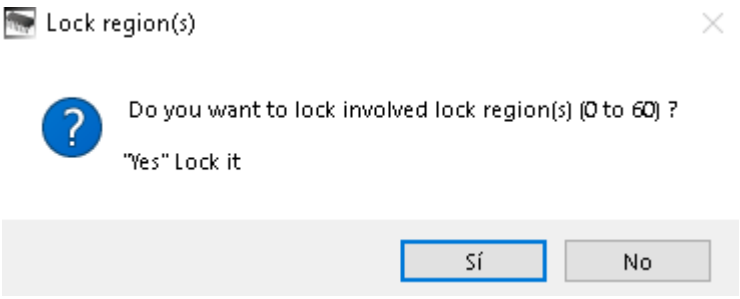


Figura 100. Ventana Emergente de Bloquear Región.

Fuente: (El autor)

9. En la sección “Scripts” seleccionamos *Boot from flash “GPNVM1”* y pulsamos el botón ejecutar. Esto se evidencia en la figura 101.

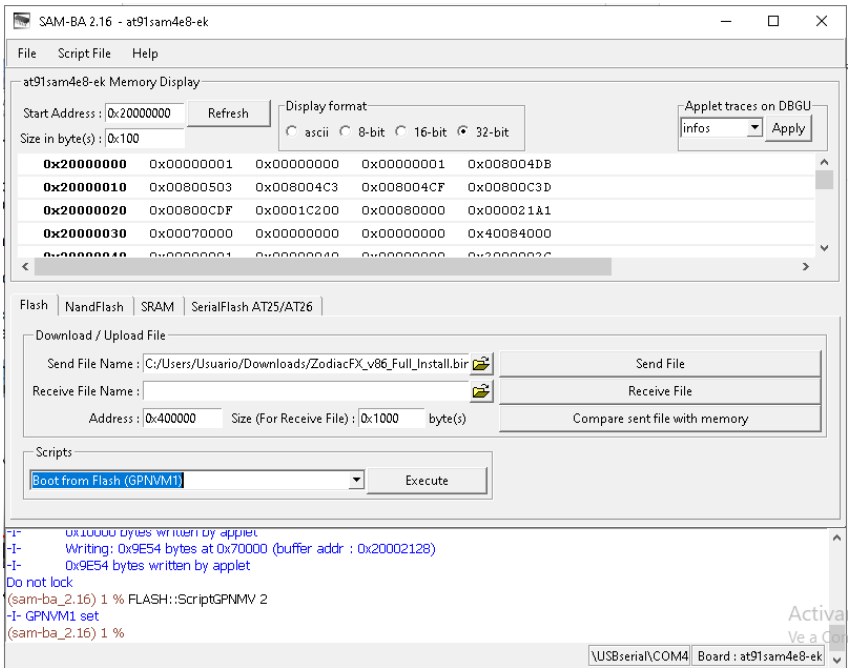


Figura 101. Selección de Script.

Fuente: (El autor)

10. Desconectamos y conectamos el cable USB nuevamente para reiniciar el dispositivo, mismo que se cargara con el firmware actualizado.

Resultados:

- **Uso de Putty**

1. Una vez conectado el cable USB del Zodiac FX a nuestra PC, es necesario a través de la consola de Ubuntu que es el sistema operativo que estamos utilizando, ver el identificador del dispositivo en serie, esto lo hacemos utilizando el comando que aparece en la figura 102.

Figura 102. Comando para identificar el Dispositivo en Serie.

Fuente: (El autor)

2. Abrimos Putty y seleccionamos la opción Serial, conectándonos a la línea serial encontrada en el paso anterior (ver figura 103).

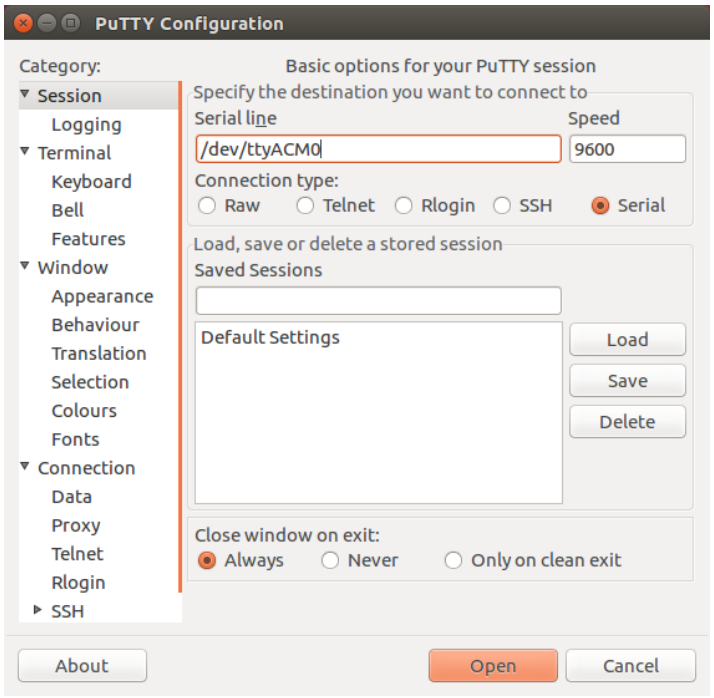


Figura 103. Abrir Putty y conexión a línea Serial.

Fuente: (El autor)

3. Abrimos la conexión y presionamos cualquier tecla en nuestro teclado para mostrar la CLI de Putty.

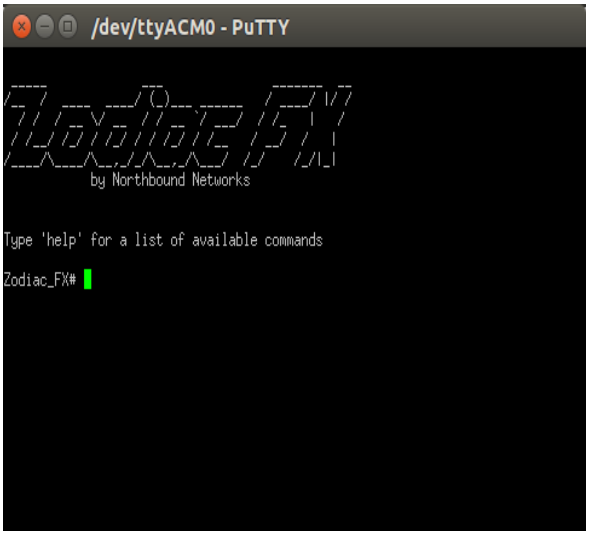


Figura 104. CLI de Putty.

Fuente: (El autor)

4. Comando ***help*** nos muestra la lista de comandos disponibles para el Zodiac FX.

Figura 105. Comandos disponibles para Zodiac FX.

Fuente: (El autor)

5. Comando ***config***

Permite ingresar al contexto config, donde se presenta una serie de subcomandos para Zodiac FX. Uno de los comandos más importantes es *show config,* que muestra la configuración actual del dispositivo (ver figura 106).

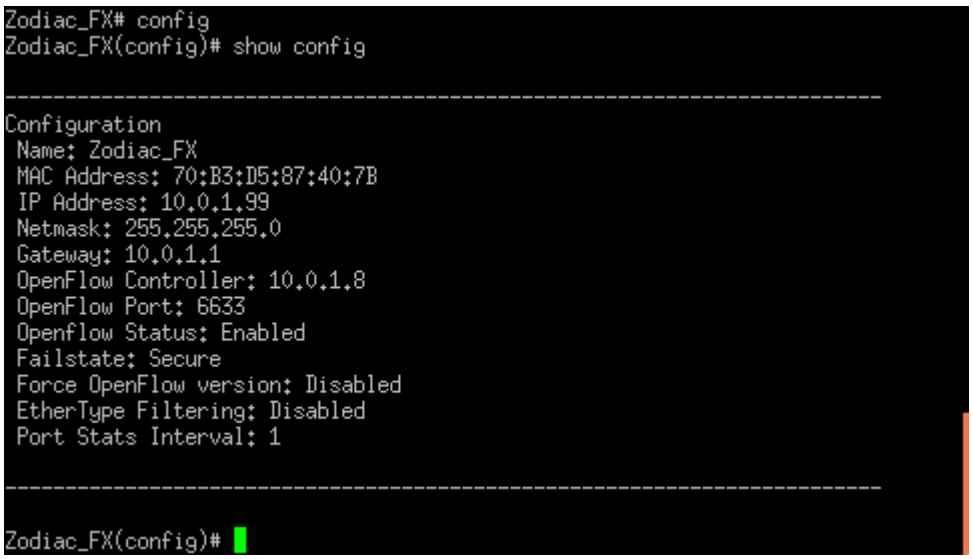


Figura 106. Comando show config.

Fuente: (El autor)

6. Son importantes también los comandos ***show status***, que muestra el estado actual del dispositivo y ***show versión,*** que muestra la versión del firmware, esto se muestra en la figura 107.

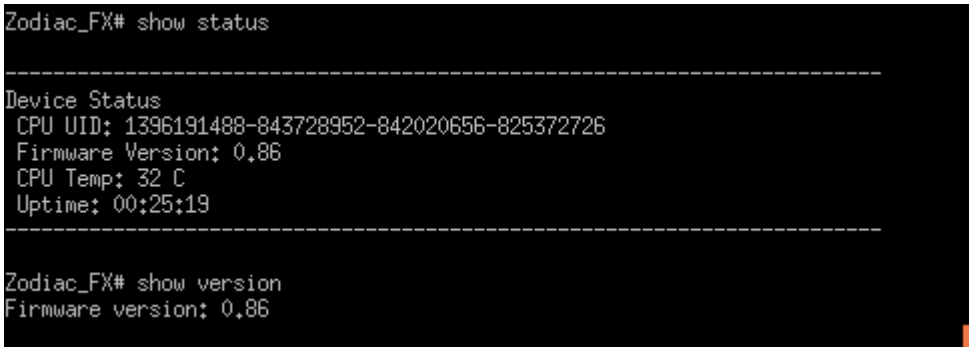


Figura 107. Estado del Dispositivo y Versión de Firmware.

Fuente: (El autor)

7. Dentro del contexto OpenFlow se encuentra el comando ***show flows*** que muestra una lista de flujos instalados actualmente. En este caso se observa que aún no tenemos flujos instalados (ver figura 108).

Figura 108. Contexto OpenFlow Comando show flows.

Fuente: (El autor)

8. Un comando bastante útil es ***exit,*** que nos permite salir de cualquier contexto dentro de la consola e Putty.

Figura 109. Comando para salir de cualquier contexto dentro de Putty.

Fuente: (El autor)

- **Uso de Interfaz Web**

La interfaz web brinda la capacidad de configurar ajustes y monitorear el funcionamiento de Zodiac FX.

Los elementos de la interfaz web se detallan a continuación:

1. Status

El botón de status permite ver el estado del dispositivo Zodiac FX, el firmare que tiene cargado y un botón de ***reset*** donde podemos devolver el dispositivo a su configuración de fábrica. Esto se muestra en la figura 110.

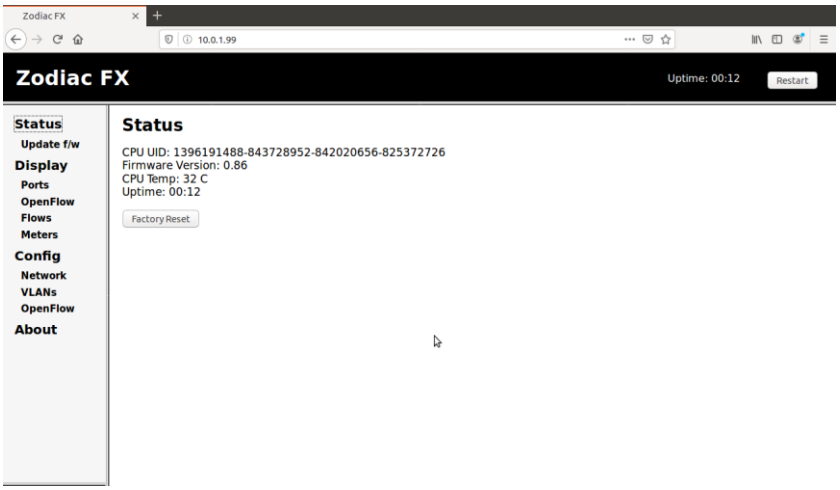


Figura 110. Status Switch Zodiac FX.

Fuente: (El autor)

2. Display

Proporciona información y estadísticas sobre diversos aspectos del Zodiac FX que se muestran a continuación:

✓ Ports

Se puede visualizar el estado de cada uno de los puertos, así como el monitoreo de las estadísticas, esto aparece en la figura 111.

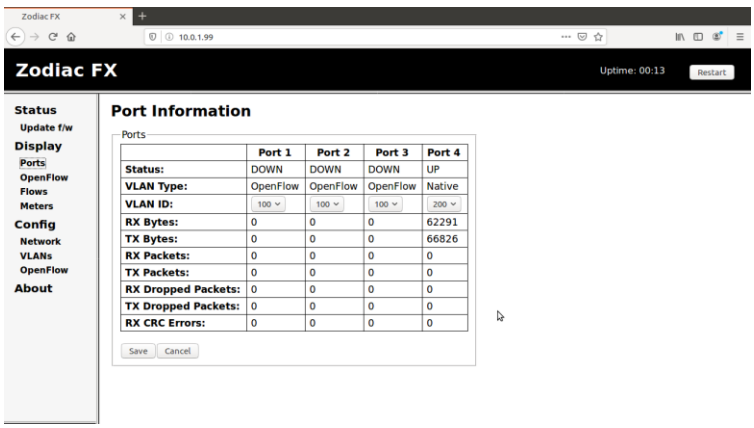


Figura 111. Estado de Puertos Zodiac FX.

Fuente: (El autor)

✓ **OpenFlow**

Muestra el estado actual y la versión de OpenFlow, así como las estadísticas de tablas de flujo (ver figura 112).

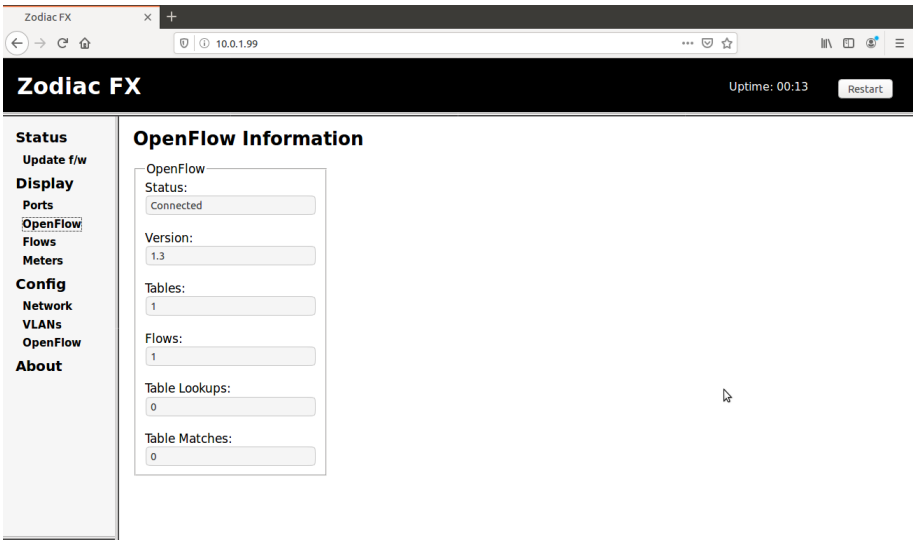


Figura 112. Información de estado y versión de OpenFlow.

Fuente: (El autor)

✓ **Flows**

Muestra los flujos actuales en la tabla de flujo junto a sus configuraciones, estos flujos se pueden borrar de forma manual con el botón “clear flows” (ver figura 113).

Figura 113. Flujos y configuración de tablas de flujo.

Fuente: (El autor)

3. Config

Esta sección muestra algunas características de configuración del Zodiac FX, y pueden ser cambiadas directamente desde la interfaz Web. Dentro de estos parámetros se puede mencionar:

- ✓ **Network**

  Muestra la configuración de red del Zodiac FX, misma que puede ser cambiada directamente desde la Interfaz web y se necesita un reinicio del dispositivo para que la configuración surta efecto (ver figura 114).

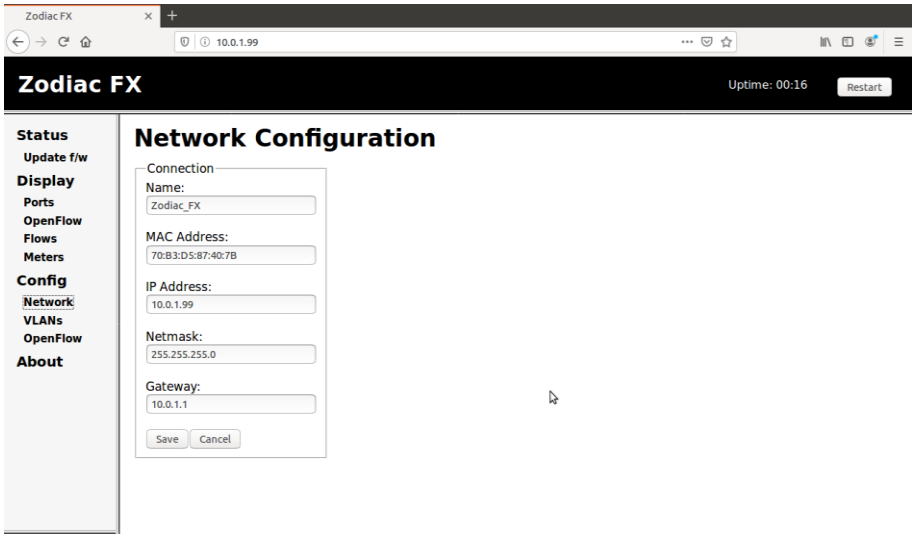


Figura 114. Configuración de Red Zodiac FX.

Fuente: (El autor)

- ✓ **VLANs**

  Las VLANs se pueden configurar de forma predeterminada como sea necesario, existen dos tipos de VLANs que pueden configurarse, las primeras para conectarse a la red y las otras para conectarse al controlador Openflow (ver figura 115).

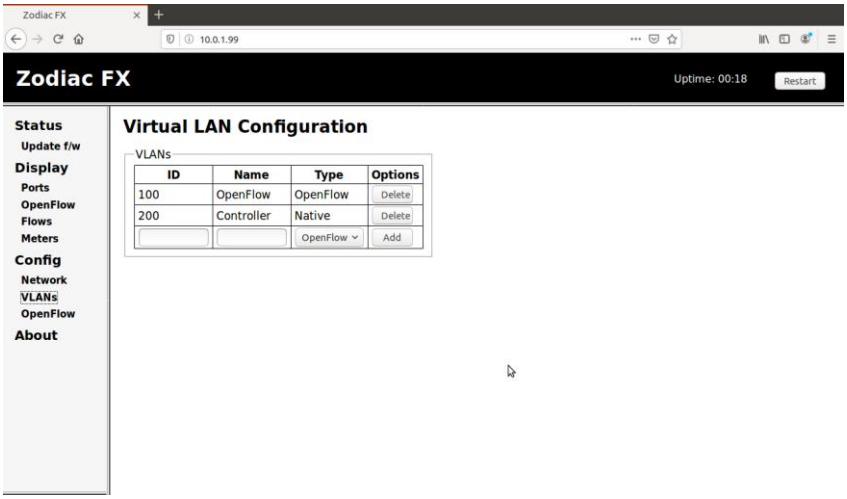


Figura 115. Configuración de VLANs Zodiac FX.

Fuente: (El autor)

✓ **OpenFlow**

En esta sección se puede configurar diversos parámetros de OpenFlow como la funcionalidad del mismo que se puede desactivar por completo, convirtiendo al dispositivo en un simple Switch, la IP y el puerto se puede modificar para permitir la comunicación entre el Switch y el controlador. Se puede también forzar la versión de OpenFlow en caso de querer cambiarla, en nuestro caso de estudio utilizaremos la versión 1.3 de OpenFlow. Esto se muestra en la figura 116.

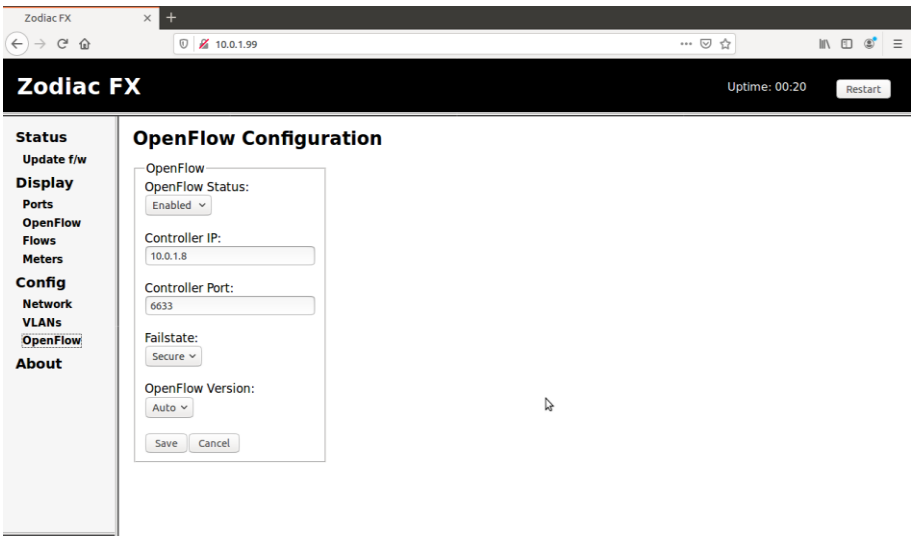


Figura 116. Configuración de Parámetros OpenFlow.

Fuente: (El autor)

Conclusiones:

- Se actualizo la versión de firmware del dispositivo Zodiac FX a su última versión utilizando para este propósito el software SAM-BA.
- Se comprendió el funcionamiento del Zodiac FX, así como de sus componentes y funciones que trae consigo.

Recomendaciones:

- Se recomienda utilizar la herramienta que más facilite la actualización de firmware, ya sea el software SAM-BA o a través de la interfaz web de Zodiac FX.
- Es recomendable utilizar herramientas como Putty que es un emulador de terminal para abrir la interfaz CLI de Zodiac FX y experimentar las funciones que este trae consigo.
- Es recomendable tener presente el puerto COM al que se está conectando nuestro Zodiac FX para poder abrirlo a través de la interfaz serial de Putty.

6.2.2 PRACTICA 2:

FUNCION SWITCH ZODIAC FX

Objetivos:

- Utilizar el Zodiac FX y darle la función de un Switch a través de la aplicación creada en Python, utilizando Ryu como controlador SDN y dos PC como host para realizar pruebas de funcionamiento.
- Utilizar la herramienta Wireshark para analizar el tráfico de paquetes entre hosts, además de la interfaz web de Zodiac FX y el emulador Putty.

Recursos utilizados:

Tabla 16. Recursos Practica 2 Zodiac FX

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7. - Switch Zodiac FX - 2 computadoras de escritorio HP (Host). - 3 cables Ethernet cat. 6	- Sistema Operativo Ubuntu 16.04 - Controlador Ryu - OpenFlow v1.3 - Aplicación Python - Putty

Fuente: (El autor)

Procedimiento:

- **Escenario de Pruebas**

Para el desarrollo de esta práctica y las practicas siguientes se utilizará un escenario compuesto por una PC donde estará funcionando el controlador Ryu y dos PC adicionales que harán la función de host, esto se muestra en la figura 117.

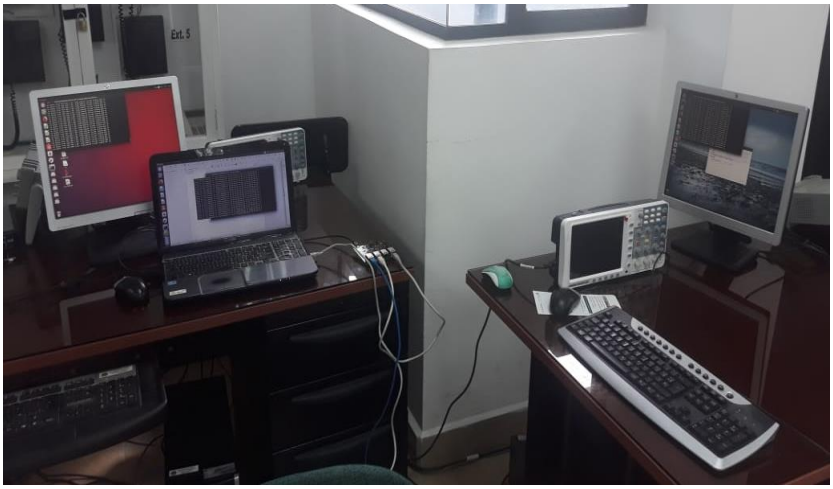


Figura 117. Escenario de Practicas Zodiac FX.

Fuente: (El autor)

- **Conexión de hosts a Zodiac FX**

Para empezar el desarrollo de la práctica es necesario conectar el cable micro USB a nuestra PC para alimentar el Zodiac FX, es necesario también conectar el puerto 4 del Zodiac a la entrada Ethernet de la PC para poder utilizar el controlador Ryu que tenemos instalada en la misma. Para conectar los hosts al Zodiac se utiliza cables Ethernet y se conecta a los puertos 1 y 2 respectivamente, como se observa en la figura 118.

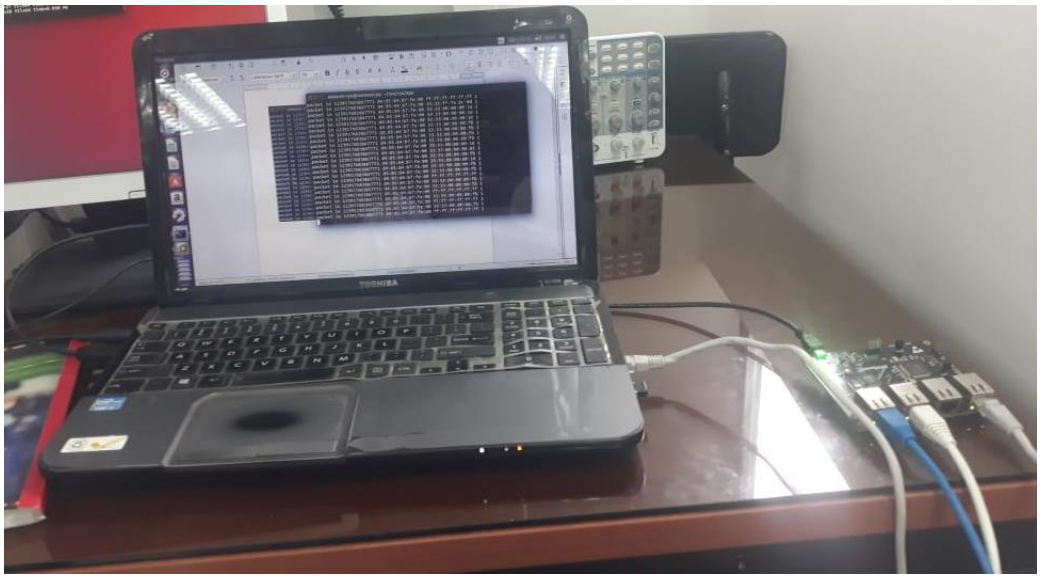


Figura 118. Conexión de Zodiac FX al Controlador y hosts.

Fuente: (El autor)

- **Topología**

La topología utilizada para el desarrollo de esta Práctica se muestra en la figura 119.

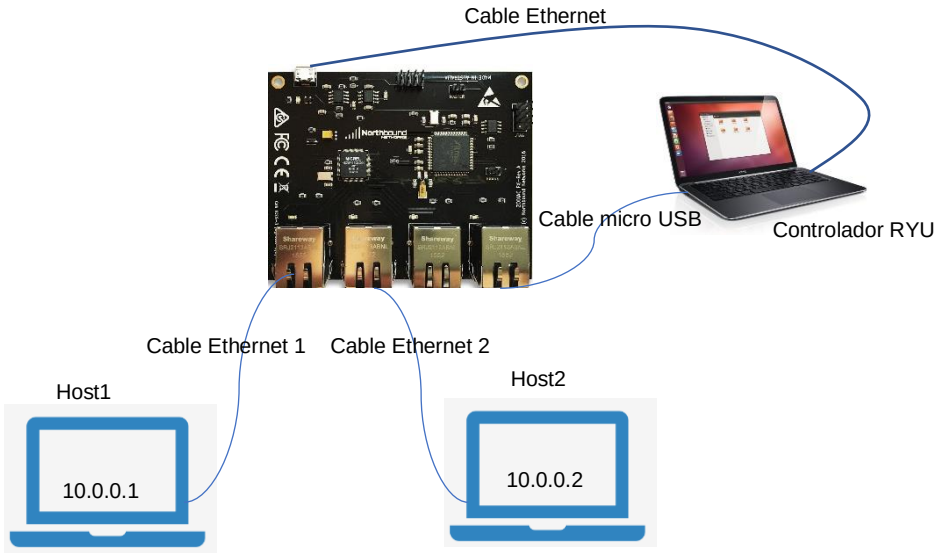


Figura 119. Topología Practica 1 Función Switch.

Fuente: (El autor)

- **Configuración de Host**

Antes de correr la Aplicación que le dará a nuestro dispositivo Zodiac FX la función de Switch, es necesario realizar configuraciones previas en cada uno de los hosts, añadiendo una IP a cada uno de ellos con la finalidad de poder establecer comunicación y analizar el tráfico de paquetes en Wireshark. La configuración de host se realiza utilizando el comando que aparece en la figura 120.

```
laboratorio@laboratorio-HP-Compaq-6000-Pro-MT-PC:~$ sudo ip addr add 10.0.1.20
/24 dev eth0
[sudo] password for laboratorio:
```

Figura 120. Configuración de host

Fuente: (El autor)

Dentro de los parámetros que se configura en este comando se encuentra la dirección IP de cada host, la máscara de red y la interfaz de comunicación.

- **Lanzamiento de la aplicación**

  Para iniciar la visualización de tráfico se debe correr la aplicación *simple_switch_13.py* proporcionada por el controlador Ryu, a través del comando *ryu-manager,* esto le dará al Zodiac FX características de Switch únicamente (ver figura 121).

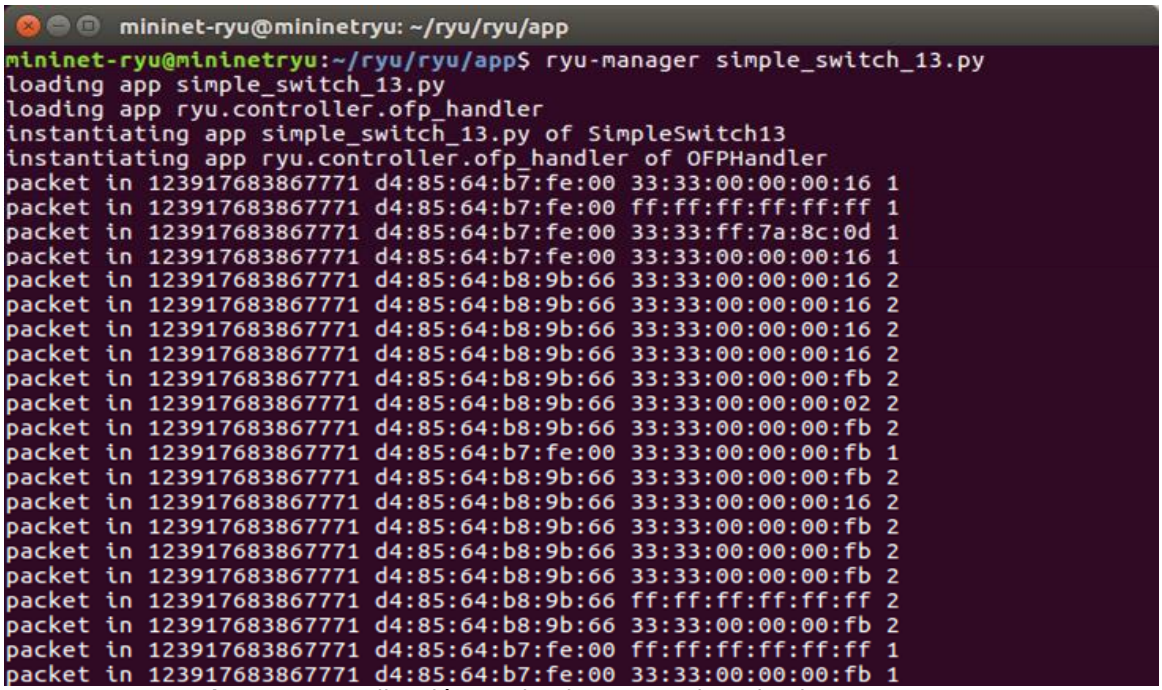


Figura 121. Aplicación corriendo conectada a dos hosts.

Fuente: (El autor)

El resultado luego de correr la aplicación se observa en la figura anterior donde se evidencia el envío de tráfico entre los hosts configurados previamente.

El código en Python de la aplicación utilizada para esta práctica se puede encontrar en el **Anexo 4** de este documento.

Resultados:

- **Resultados en Interfaz Web**

  Como se mencionó en la práctica anterior es posible ver el resultado de la configuración que se dé al Zodiac FX, accediendo a la interfaz web utilizando la IP 10.0.1.99 que es la dirección que tiene configurada nuestro Zodiac FX.

Las figuras 122 y 123 muestran los flujos instalados en el Zodiac FX, la primera corresponde al flujo generado por el Controlador Ryu en el Puerto 4, y la siguiente muestra los flujos producidos por los puertos 1 y 2 conectados a cada uno del host.

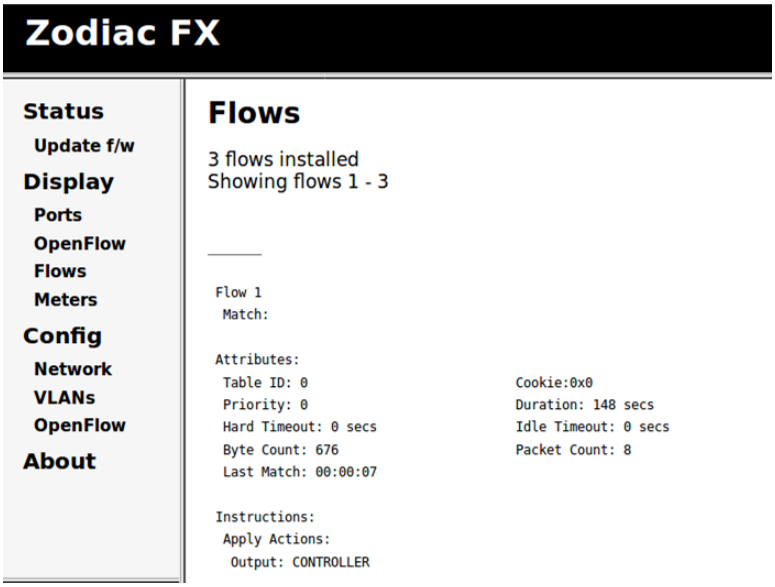


Figura 122. Flujo creado por el Puerto 4 conectado al Controlador Ryu.

Fuente: (El autor)

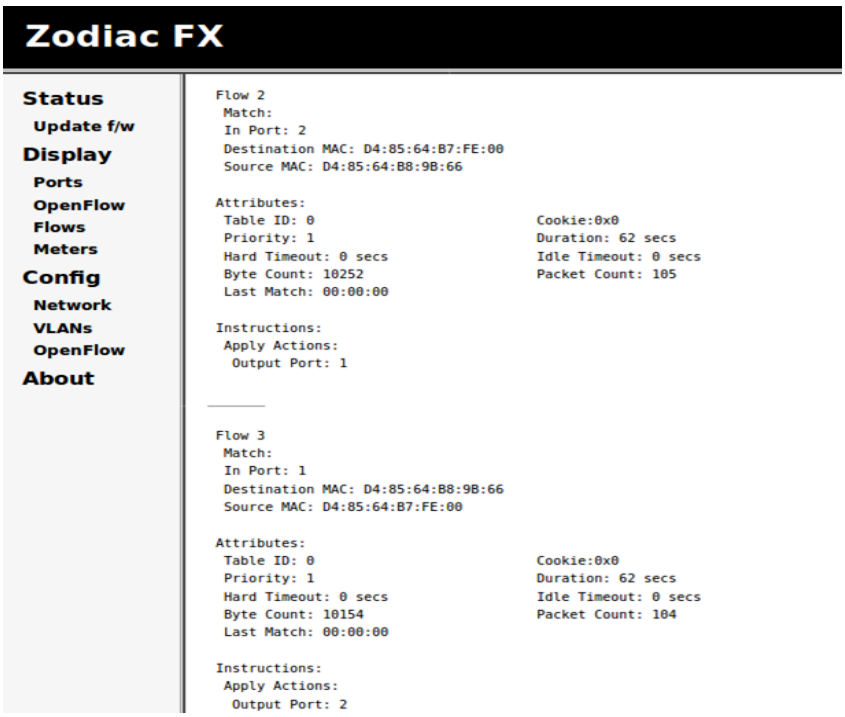


Figura 123. Flujos producidos por los puertos 1 y 2 conectados a los hosts.

Fuente: (El autor)

Se puede evidenciar además el estado de los puertos utilizados, la cantidad de Bytes transmitidos durante el tiempo de conexión entre los hosts y el controlador (ver figura 124).

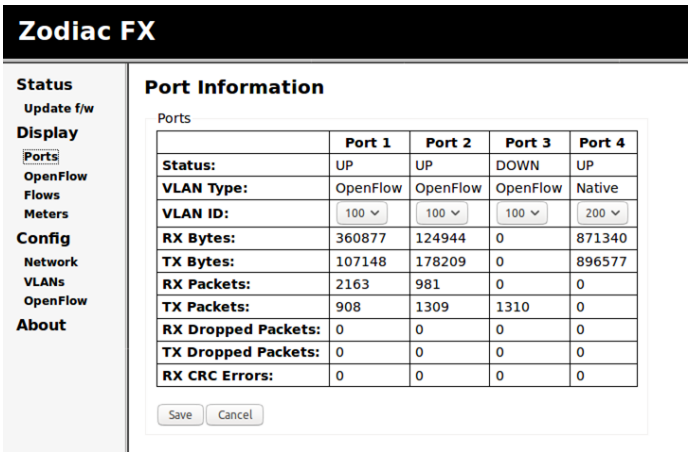


Figura 124. Estado de los Puertos conectados a los hosts.

Fuente: (El autor)

- **Resultados en Putty**

  Se puede evidenciar resultados similares a los anteriores utilizando la herramienta Putty, las figuras 125 y 126 muestran los flujos creados por Zodiac FX y el estado de los puertos respectivamente.

```
Zodiac_FX(openflow)# show flows
------------------------------------------------------------------------

Flow 1
 Match:
 Attributes:
  Table ID: 0                          Cookie:0x0
  Priority: 0                          Duration: 266 secs
  Hard Timeout: 0 secs                 Idle Timeout: 0 secs
  Byte Count: 746                      Packet Count: 9
  Last Match: 00:01:44
 Instructions:
  Apply Actions:
   Output: CONTROLLER

Flow 2
 Match:
  In Port: 2
  Destination MAC: D4:85:64:B7:FE:00
  Source MAC: D4:85:64:B8:9B:66
 Attributes:
  Table ID: 0                          Cookie:0x0
  Priority: 1                          Duration: 180 secs
  Hard Timeout: 0 secs                 Idle Timeout: 0 secs
  Byte Count: 33184                    Packet Count: 339
  Last Match: 00:00:00
 Instructions:
  Apply Actions:
   Output Port: 1

Flow 3
 Match:
  In Port: 1
  Destination MAC: D4:85:64:B8:9B:66
  Source MAC: D4:85:64:B7:FE:00
 Attributes:
  Table ID: 0                          Cookie:0x0
  Priority: 1                          Duration: 180 secs
  Hard Timeout: 0 secs                 Idle Timeout: 0 secs
  Byte Count: 33086                    Packet Count: 338
  Last Match: 00:00:00
 Instructions:
  Apply Actions:
   Output Port: 2
```

Figura 125. Flujos creados en Zodiac FX mostrados en Putty.

Fuente: (El autor)

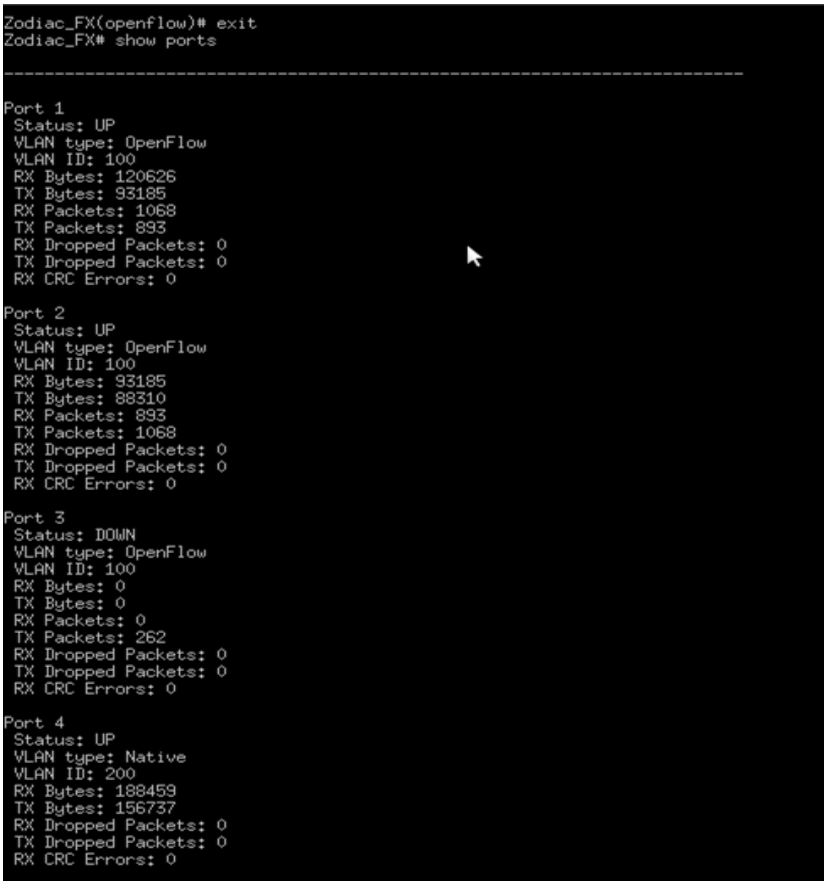


Figura 126. Estado de los puertos mostrados en Putty.

Fuente: (El autor)

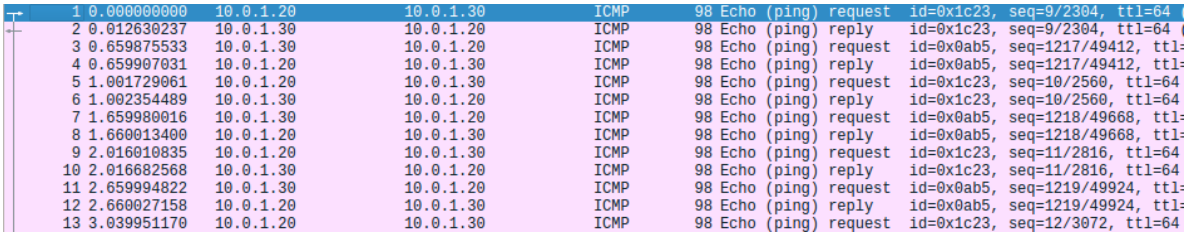
- **Resultados en Wireshark**

Es posible utilizar la herramienta Wireshark para analizar el tráfico de paquetes entre hosts y controlador. Las figuras 127 y 128 muestran el tráfico del controlador al correr la aplicación de Switch, y también el tráfico entre los hosts respectivamente al hacer una prueba de Eco.

10	8.769025162	10.0.1.99	10.0.1.8	TCP	60 49725 → 6633 [ACK] Seq=25 Ack=25 Win=2064 Len=0
11	11.778201324	10.0.1.99	10.0.1.8	OpenFlow	62 Type: OFPT_ECHO_REQUEST
12	11.778987196	10.0.1.8	10.0.1.99	OpenFlow	62 Type: OFPT_ECHO_REPLY
13	12.028143819	10.0.1.99	10.0.1.8	TCP	60 49725 → 6633 [ACK] Seq=33 Ack=33 Win=2056 Len=0
14	14.759214896	10.0.1.99	10.0.1.8	OpenFlow	62 Type: OFPT_ECHO_REQUEST
15	14.760000491	10.0.1.8	10.0.1.99	OpenFlow	62 Type: OFPT_ECHO_REPLY
16	14.783080569	10.0.1.99	10.0.1.8	TCP	60 49725 → 6633 [ACK] Seq=41 Ack=41 Win=2048 Len=0
17	17.792258262	10.0.1.99	10.0.1.8	OpenFlow	62 Type: OFPT_ECHO_REQUEST
18	17.793047489	10.0.1.8	10.0.1.99	OpenFlow	62 Type: OFPT_ECHO_REPLY
19	18.042173825	10.0.1.99	10.0.1.8	TCP	60 49725 → 6633 [ACK] Seq=49 Ack=49 Win=2040 Len=0
20	20.773334436	10.0.1.99	10.0.1.8	OpenFlow	62 Type: OFPT_ECHO_REQUEST
21	20.774041983	10.0.1.8	10.0.1.99	OpenFlow	62 Type: OFPT_ECHO_REPLY
22	20.797098918	10.0.1.99	10.0.1.8	TCP	60 49725 → 6633 [ACK] Seq=57 Ack=57 Win=2032 Len=0

Figura 127. Trafico generado por el Controlador.

Fuente: (El autor)



1	0.000000000	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) request id=0x1c23, seq=9/2304, ttl=64
2	0.012630237	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) reply id=0x1c23, seq=9/2304, ttl=64
3	0.659875533	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) request id=0x0ab5, seq=1217/49412, ttl=
4	0.659907031	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) reply id=0x0ab5, seq=1217/49412, ttl=
5	1.001729061	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) request id=0x1c23, seq=10/2560, ttl=64
6	1.002354489	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) reply id=0x1c23, seq=10/2560, ttl=64
7	1.659980016	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) request id=0x0ab5, seq=1218/49668, ttl=
8	1.660013400	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) reply id=0x0ab5, seq=1218/49668, ttl=
9	2.016010835	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) request id=0x1c23, seq=11/2816, ttl=64
10	2.016682568	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) reply id=0x1c23, seq=11/2816, ttl=64
11	2.659994822	10.0.1.30	10.0.1.20	ICMP	98	Echo (ping) request id=0x0ab5, seq=1219/49924, ttl=
12	2.660027158	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) reply id=0x0ab5, seq=1219/49924, ttl=
13	3.039951170	10.0.1.20	10.0.1.30	ICMP	98	Echo (ping) request id=0x1c23, seq=12/3072, ttl=64

Figura 128. Trafico entre Host1 y Host 2 Practica 2 Zodiac FX.

Fuente: (El autor)

Conclusiones:

- Se utilizó el Switch Zodiac FX y se le configuró la función de Switch a través de la aplicación creada en Python, utilizando Ryu como controlador SDN y dos PC como host para realizar pruebas de funcionamiento.
- Se utilizó la herramienta Wireshark para analizar el tráfico de paquetes entre hosts, además de la interfaz web de Zodiac FX y el emulador Putty para observar el estado del dispositivo durante el funcionamiento de la aplicación.

Recomendaciones:

- Se recomienda utilizar las herramientas de tráfico de paquetes como Wireshark, además de la interfaz web de Zodiac y el emulador de terminal Putty para poder observar el comportamiento de la red, durante su ejecución.
- Se recomienda utilizar como Host computadores cargados con Sistema Operativo Ubuntu con la finalidad de facilitar la configuración y funcionamiento de la red, ya que el Controlador está funcionando bajo este sistema Operativo.

6.2.3 PRACTICA 3:

FUNCION ROUTER ZODIAC FX

Objetivos:

- Utilizar el Switch Zodiac FX y darle la función de Router a través de una aplicación creada en Python, utilizando Ryu como controlador SDN y dos PC como host para realizar pruebas de funcionamiento.
- Utilizar la herramienta Wireshark para analizar el tráfico de paquetes entre hosts.

Recursos utilizados:

Tabla 17. Recursos Practica 3 Zodiac FX

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7. - Switch Zodiac FX - 3 computadoras de escritorio HP (Host). - 4 cables Ethernet cat. 6	- Sistema Operativo Ubuntu 16.04 - Controlador Ryu - OpenFlow v1.3 - Aplicación Python - Putty

Fuente: (El autor)

Procedimiento:

- **Topología**

  La topología que se pretende implementar en esta práctica se muestra en la figura 129.

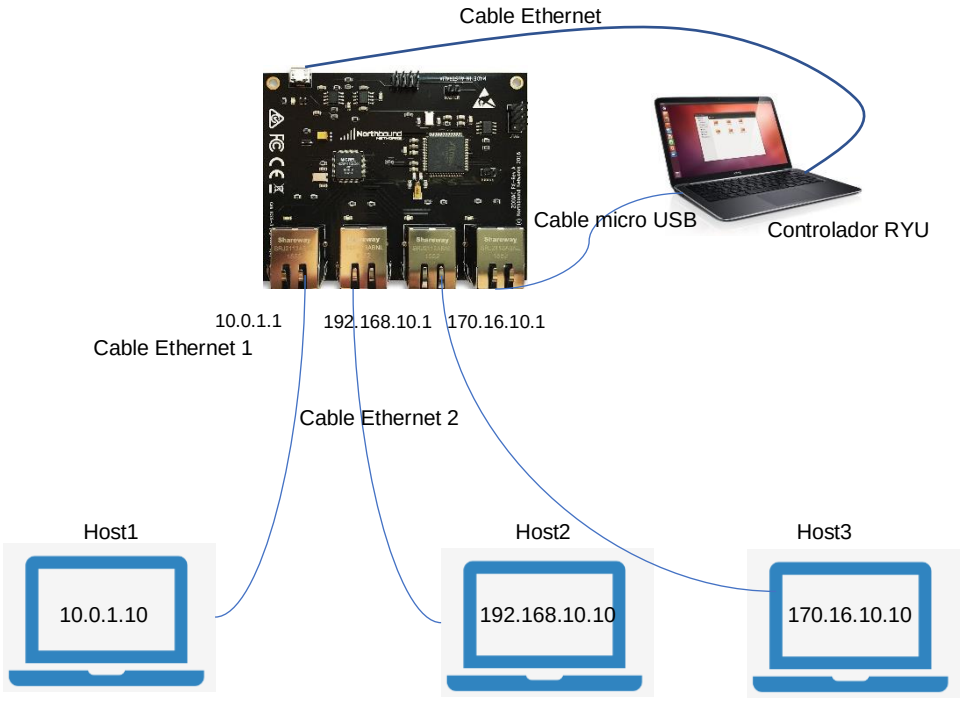


Figura 129. Topología Práctica 3 Función Router.

Fuente: (El autor)

- **Configuración de hosts**

  La configuración de Host se realiza en los dispositivos que formaran parte de la red, como lo muestra la figura 130.

```
laboratorio@laboratorio-HP-Compaq-6000-Pro-MT-PC:~$ sudo ip addr add 172.16.10
.10/24 dev eth0
```

Figura 130. Configuración de Hosts Practica 3 Zodiac FX.

Fuente: (El autor)

- **Lanzamiento de Aplicación**

  Ejecutamos la Aplicación ***rest_router.py*** que le dará al Zodiac FX características de ruteo, además es necesario lanzar la aplicación ***simple_switch_13.py*** de forma simultánea con la finalidad de obtener las tablas MAC de cada uno de los hosts conectados al Switch y aplicar posteriormente el ruteo. Esto se muestra en la figura 131.

```
geovanny@ubuntu-Ryu:~/ryu/ryu/app$ ryu-manager simple_switch_13.py rest_router.p
y
loading app simple_switch_13.py
loading app rest_router.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app rest_router.py of RestRouterAPI
(22262) wsgi starting up on http://0.0.0.0:8080
[RT][INFO] switch_id=000070b3d587407b: Set SW config for TTL error packet in.
[RT][INFO] switch_id=000070b3d587407b: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=000070b3d587407b: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=000070b3d587407b: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=000070b3d587407b: Start cyclic routing table update.
[RT][INFO] switch_id=000070b3d587407b: Join as router.
```

Figura 131. Lanzamiento de Aplicación para función Router.

Fuente: (El autor)

Luego de lanzar la aplicación observamos que nuestro Zodiac FX ha sido reconocido y asignado un ***id*** formado por un número de16 cifras, dónde se encuentra la MAC del dispositivo. Es necesario añadir algunas líneas de comando REST que indicaran los Gateway de cada red para establecer comunicación entre hosts de igual forma como se realizó con el simulador Mininet, lo único que se debe considerar es el ***id*** del Zodiac FX. Este se muestra en las siguientes líneas:

curl -X POST -d '{"gateway": "10.0.1.1"}' http://localhost:8080/router/000070b3d587407b

curl -X POST -d '{"gateway": "192.168.10.1"}' http://localhost:8080/router/000070b3d587407b

curl -X POST -d '{"gateway": "170.16.10.1"}' http://localhost:8080/router/000070b3d587407b

El código en Python utilizado para el desarrollo de esta Práctica se muestra en el **Anexo 5** de este documento.

Resultados:

- **Prueba de Eco (ping) entre hosts**

  Las pruebas de eco para cada host se evidencian en las figuras 132, 133 y 134 respectivamente.

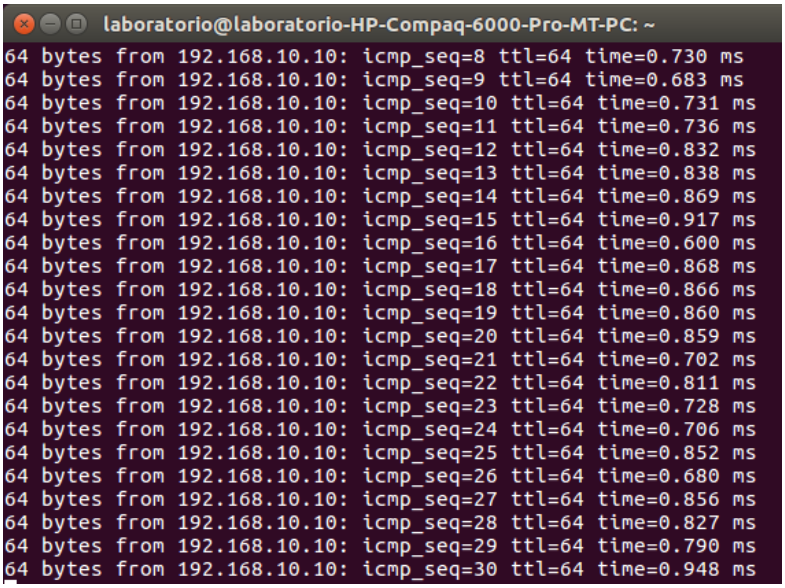


Figura 132. Ping de host 2 a host 1.

Fuente: (El autor)

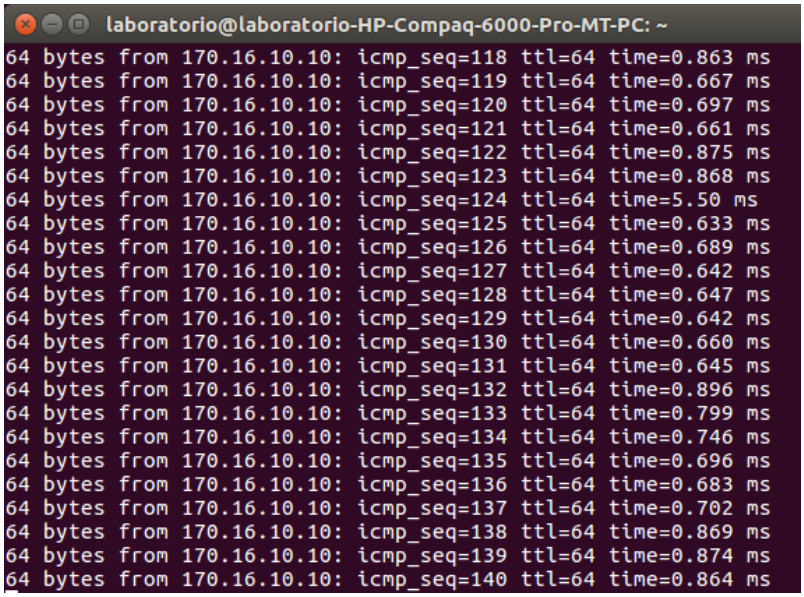


Figura 133. Ping de host 3 a host 1.

Fuente: (El autor)

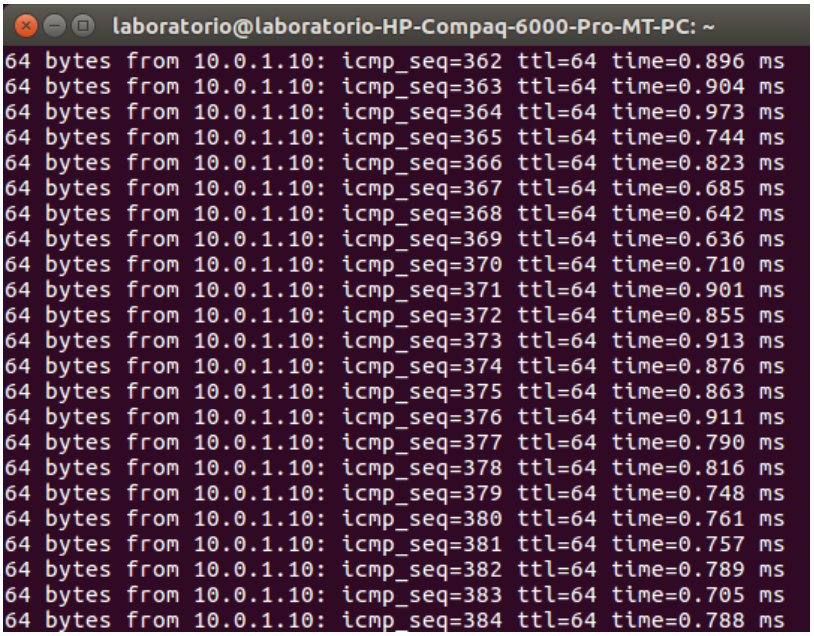


Figura 134. Ping de host 1 a host 3.

Fuente: (El autor)

- **Resultados Wireshark**

  Se analiza también el tráfico generado por la prueba de eco entre hosts utilizando la herramienta Wireshark (ver figura 135).

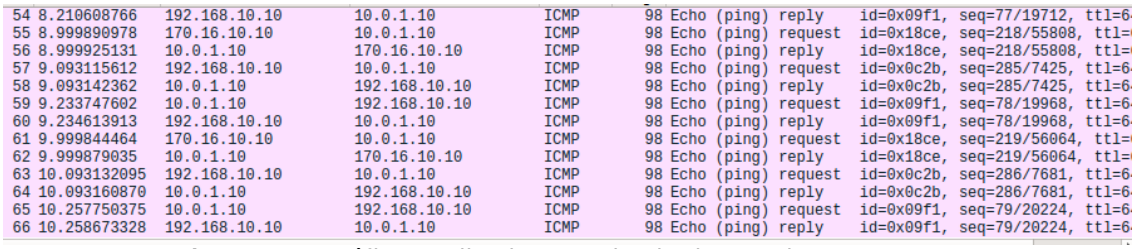


Figura 135. Tráfico analizado con Wireshark entre hosts.

Fuente: (El autor)

Se puede además realizar un ping a los Gateway de las diferentes redes, observando los resultados en el controlador, mas no en los hosts, ya que considerando que es una SDN, el controlador es el que maneja todos los eventos. Esto se muestra en la figura 136.

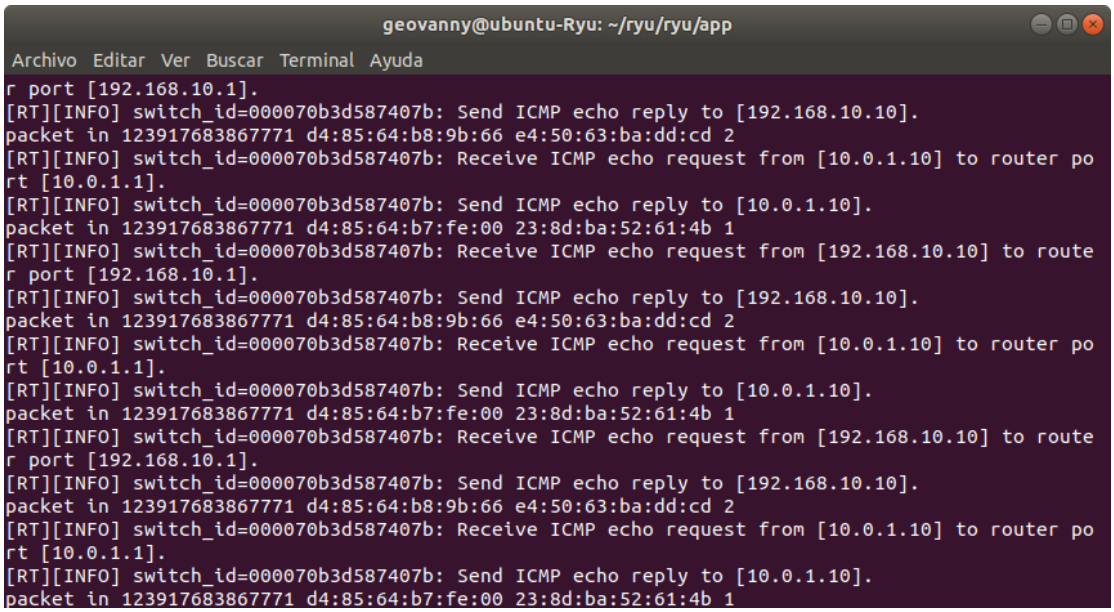


Figura 136. Pruebas de eco a los Gateway de la red mostradas en el controlador.

Conclusiones:

- Se utilizó el Switch Zodiac FX y se le otorgó la función de Router a través de una aplicación creada en Python, utilizando Ryu como controlador SDN y dos PC como host para realizar pruebas de funcionamiento.
- Se utilizó la herramienta Wireshark para analizar el tráfico de paquetes entre hosts.

Recomendaciones:

- Se recomienda utilizar las herramientas de tráfico de paquetes como Wireshark para el análisis del estado de la red.
- Es recomendable verificar el estado de red de los Hosts utilizados para llevar a cabo el desarrollo de la práctica con la finalidad que no exista conflicto duranta la ejecución de la misma.

6.2.4 PRACTICA 4:

FUNCION FIREWALL ZODIAC FX

Objetivos:

- Utilizar el Switch Zodiac FX para configurarle la función de Firewall a través de reglas implementadas en una aplicación Python y de esta forma proteger nuestra red a nivel de usuarios finales.
- Utilizar la herramienta Wireshark para analizar el tráfico de paquetes entre hosts.

Recursos utilizados:

Tabla 18. Recursos Practica 4 Zodiac FX

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7. - Switch Zodiac FX - 3 computadoras de escritorio HP (Host). - 4 cables Ethernet cat. 6	- Sistema Operativo Ubuntu 16.04 - Controlador Ryu - OpenFlow v1.3 - Aplicación Python - Putty

Fuente: (El autor)

Procedimiento:

- **Topología**

  Para el desarrollo de esta Práctica se utiliza la Topología que aparece en la siguiente figura:

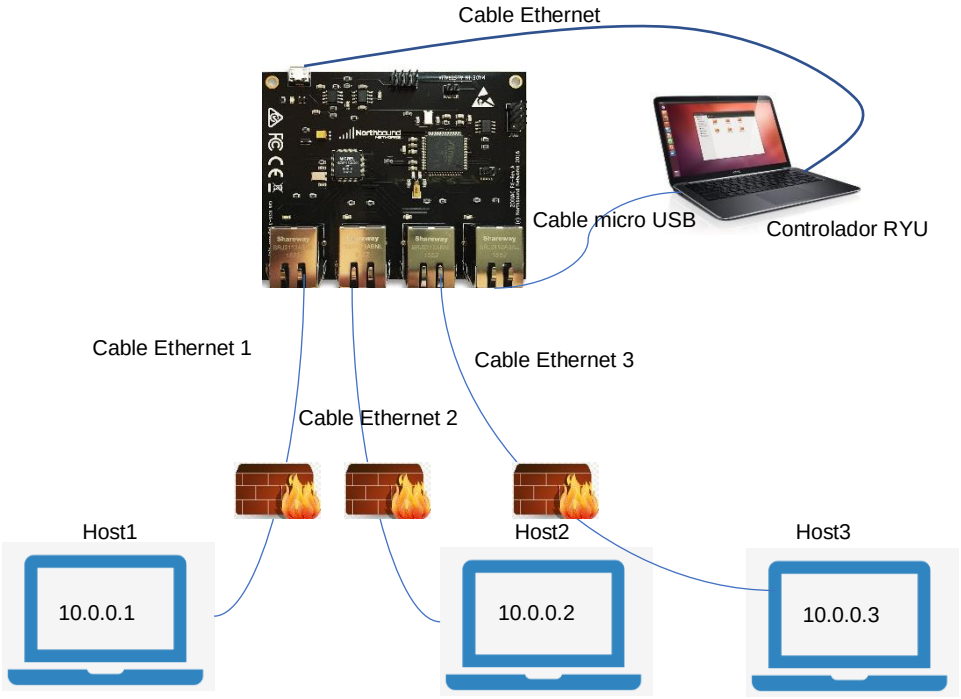


Figura 137. Topología Practica 4 Función Firewall.

Fuente: (El autor)

- **Lanzamiento de la Aplicación**

  Para lanzar la aplicación que dará a nuestro Zodiac FX la función de Firewall se sigue el mismo procedimiento mostrado en prácticas anteriores, esto se evidencia en la figura 138.

```
mininet-ryu@mininetryu:~/ryu/ryu/app$ ryu-manager ryu.app.ofctl_rest rest_firewa
ll.py
loading app ryu.app.ofctl_rest
loading app rest_firewall.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(4745) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=000070b3d587407b: Join as firewall.
```

Figura 138. Inicio de Controlador cargando la Función Firewall.

Fuente: (El autor)

El código en Python utilizado para esta práctica se muestra en el **Anexo 6** de este documento.

- **Configuración de Reglas y Pruebas de funcionamiento**

Es necesario habilitar la función de Zodiac FX antes de configurar cualquier regla, esto se hace utilizando el comando que muestra la figura 139.

```
root@mininetryu:/home/mininet-ryu# curl -X PUT http://localhost:8080/firewall/mo
dule/enable/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "deta
ils": "firewall running."}}]root@mininetryu:/home/mininet-ryu#
```

Figura 139. Habilitar función Firewall.

Fuente: (El autor)

Si establecemos un ping entre Host 1 y Host 2 la comunicación se bloqueará ya que la configuración de reglas de permiso de acceso aún no se encuentra establecida, este resultado se muestra en la salida del controlador de la siguiente figura.

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:
02',ethertype=2048,src='00:00:00:00:00:01'), ipv4(csum=7832,dst='10.0.0.2',flags
=2,header_length=5,identification=2063,offset=0,option=None,proto=1,src='10.0.0.
1',tos=0,total_length=84,ttl=64,version=4), icmp(code=0,csum=24072,data=echo(dat
a='lT(^\x00\x00\x00\x00\xe6\x8b\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\
x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567',id=22483,seq=1
9),type=8)
```

Figura 140. Bloqueo de Comunicación entre Host 1 y Host 2.

Fuente: (El autor)

Se añade las reglas que cumplirá el firewall, en este caso habilitaremos únicamente el tráfico ICMP a los hosts, bloqueando cualquier otro tipo de tráfico. Esto se muestra en la siguiente figura.

```
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.1/32", "n
w_dst": "10.0.0.2/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules
/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=1"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.2/32", "n
w_dst": "10.0.0.1/32", "nw_proto": "ICMP"}' http://localhost:8080/firewall/rules
/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=2"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.2/32", "n
w_dst": "10.0.0.3/32"}' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=3"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.3/32", "n
w_dst": "10.0.0.2/32"}' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=4"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.1/32", "n
w_dst": "10.0.0.3/32"}' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=5"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d '{"nw_src": "10.0.0.3/32", "n
w_dst": "10.0.0.1/32"}' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=6"}]}]root@mininetryu:/home/mininet-ryu#
```

Figura 141. Reglas para habilitar ICMP Practica 4.

Fuente: (El autor)

Resultados:

Se puede establecer un ping entre todos los Hosts que forman parte de la red. esto se evidencia en las figuras 142 y 143.

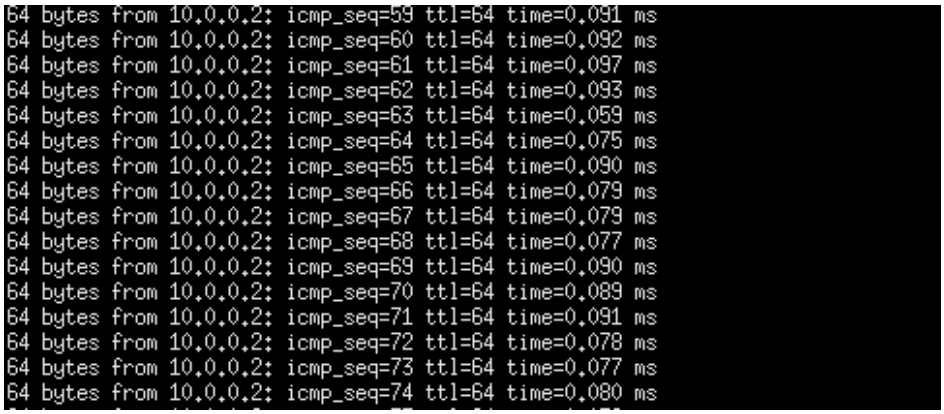


Figura 142. Ping entre Host 1 y Host 2 Práctica 4 Zodiac FX.

Fuente: (El autor)

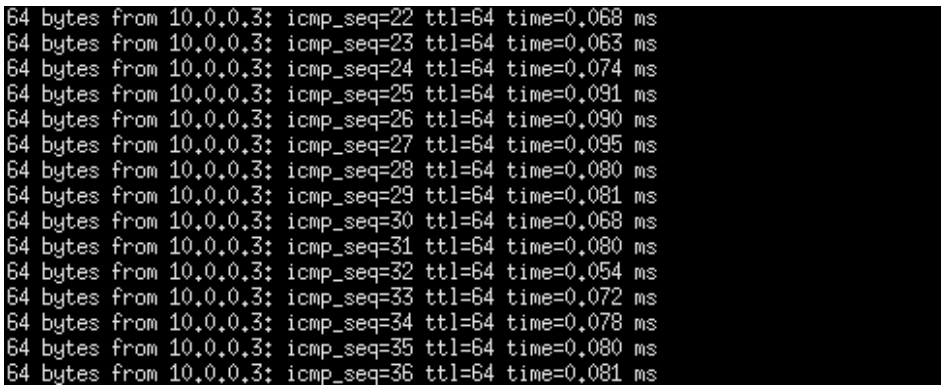


Figura 143. Ping entre Host 1 y Host 3 Práctica 4 Zodiac FX.

Fuente: (El autor)

De igual forma que en casos anteriores se puede analizar el tráfico utilizando la herramienta WireShark como lo muestra las siguientes figuras:

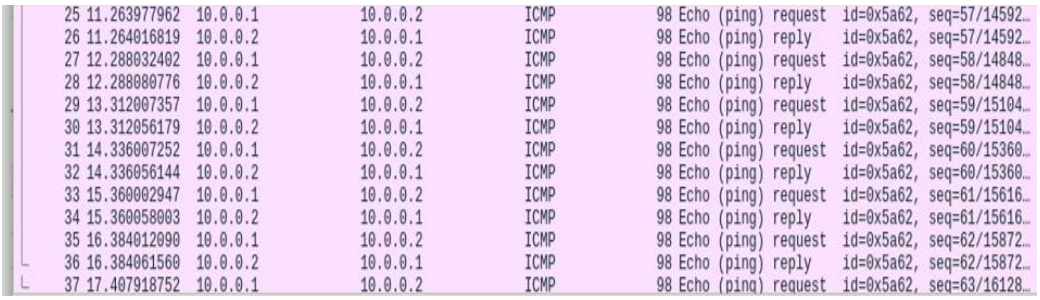


Figura 144. Tráfico en WireShark entre Host 1 y Host 2 Práctica 4 Zodiac FX.

Fuente: (El autor)

51	24.576068889	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=40/10240…
52	24.576113347	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=40/10240…
53	25.600083000	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=41/10496…
54	25.600127892	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=41/10496…
55	26.624094474	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=42/10752…
56	26.624139093	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=42/10752…
57	27.648118934	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=43/11008…
58	27.648167849	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=43/11008…
59	28.672140068	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=44/11264…
60	28.672189458	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=44/11264…
61	29.695995559	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0x5bf1, seq=45/11520…
62	29.696019695	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0x5bf1, seq=45/11520…

Figura 145. Tráfico en WireShark entre Host 1 y Host 3 Práctica 4 Zodiac FX.

Fuente: (El autor)

Ahora procedemos a bloquear el tráfico ICMP entre Host 2 y Host 3 estableciendo las reglas que aparecen en la figura 146. Cabe mencionar que el bloqueo tiene que ser bidireccional.

```
root@mininetryu:/home/mininet-ryu# curl -X POST -d  '{"nw_src": "10.0.0.2/32", "
nw_dst": "10.0.0.3/32", "nw_proto": "ICMP", "actions": "DENY", "priority": "10"}
' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=7"}]}]root@mininetryu:/home/mininet-ryu#
root@mininetryu:/home/mininet-ryu# curl -X POST -d  '{"nw_src": "10.0.0.3/32", "
nw_dst": "10.0.0.2/32", "nw_proto": "ICMP", "actions": "DENY", "priority": "10"}
' http://localhost:8080/firewall/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "det
ails": "Rule added. : rule_id=8"}]}]root@mininetryu:/home/mininet-ryu#
```

Figura 146. Bloqueo de tráfico ICMP entre Host 2 y Host 3.

Fuente: (El autor)

Luego de haber configurado estas reglas tendremos una topología como la que se muestra en la figura 147.

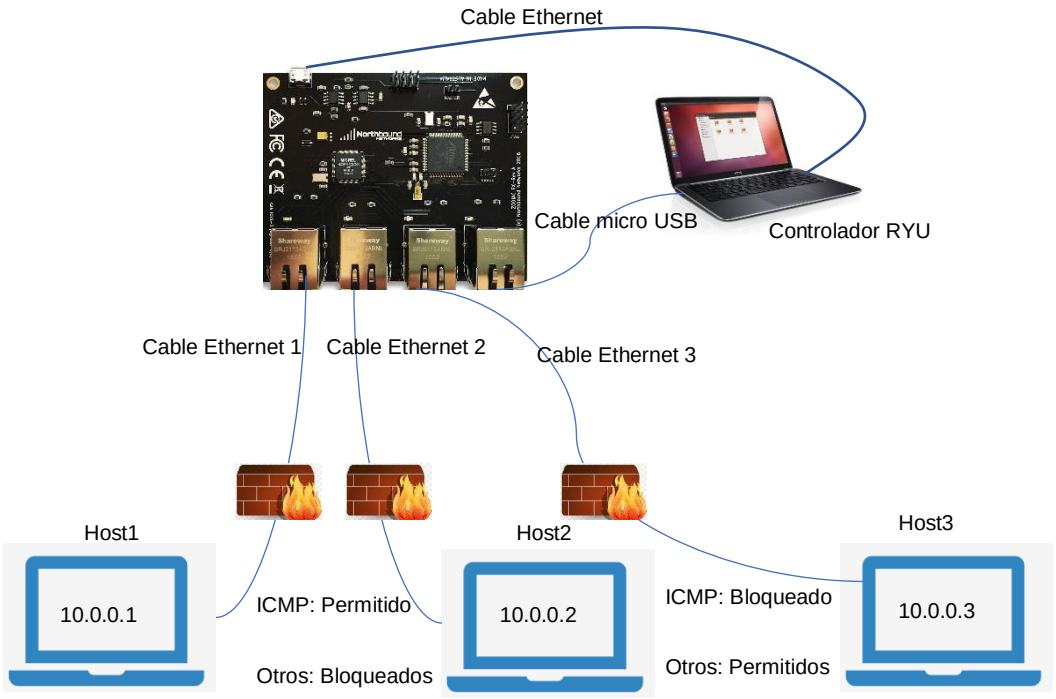


Figura 147. Topología luego de añadir reglas al Firewall.

Fuente: (El autor)

El firewall bloquea del Host1 al Host2 todo lo que no sea tráfico ICMP, por ejemplo, si se ejecuta wget del Host 1 al Host 2 el controlador mostrara que los paquetes fueron bloqueados (ver figura 148).

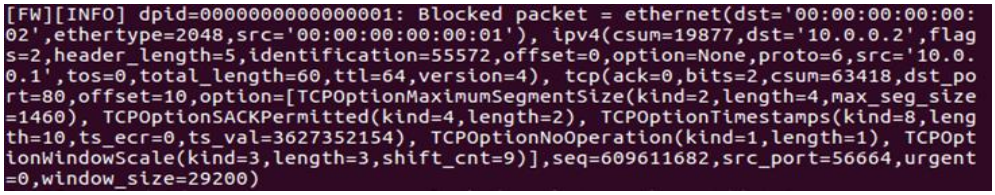


Figura 148. Bloqueo de paquetes entre Host 2 y Host 3 Practica 4.

Fuente: (El autor)

Entre el Host 2 y el Host 3 los paquetes que no sean ICMP pueden comunicarse, por ejemplo, si se ejecuta ssh del Host 2 al Host 3, no se genera algún registro en el controlador de que los mensajes fueron Bloqueados. Lo único que se muestra es que ssh no se encuentra instalado en el Host 3 (ver figura 149).

Figura 149. ssh entre Host 2 y Host 3 Practica 4 Zodiac FX.

Fuente: (El autor)

Si se registra un ping de Host 2 a Host 3 se muestra un registro de bloqueo de paquetes.

Figura 150. Bloqueo de Paquetes ICMP entre Host 2 y Host 3.

Fuente: (El autor)

Ahora eliminamos las reglas de Firewall entre Host 2 y Host 3, de tal forma que se permita todo el tráfico entre ellos.

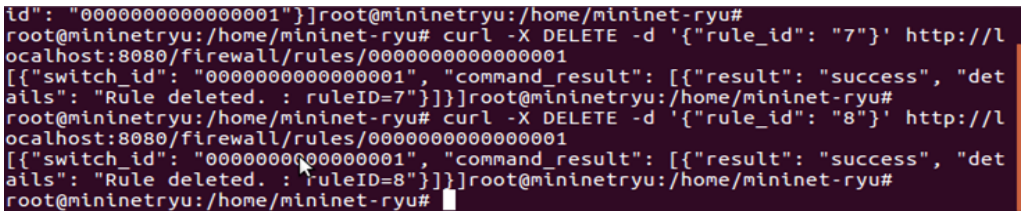


Figura 151. Eliminar Reglas Firewall entre Host 2 y Host 3.

Fuente: (El autor)

En el controlador también se muestra el resultado de las reglas eliminadas.

```
127.0.0.1 - - [22/Jan/2020 09:53:09] "DELETE /firewall/rules/0000000000000001 HT
TP/1.1" 200 241 0.012559
(21598) accepted ('127.0.0.1', 47566)
127.0.0.1 - - [22/Jan/2020 09:53:43] "DELETE /firewall/rules/0000000000000001 HT
TP/1.1" 200 241 0.006500
```

Figura 152. Resultados en el Controlador de Reglas de Firewall eliminadas.

Fuente: (El autor)

Si analizamos los paquetes en la herramienta Wireshark, observamos que todo el tráfico desde Host 2 a Host 3 está permitido. Esto se muestra en la figura 153.

117 752.165032917	10.0.0.2	10.0.0.3	ICMP	98 Echo (ping) request id=0x5f2f, seq=26/6656,…
118 752.165077219	10.0.0.3	10.0.0.2	ICMP	98 Echo (ping) reply id=0x5f2f, seq=26/6656,…
119 753.189125357	10.0.0.2	10.0.0.3	ICMP	98 Echo (ping) request id=0x5f2f, seq=27/6912,…
120 753.189174411	10.0.0.3	10.0.0.2	ICMP	98 Echo (ping) reply id=0x5f2f, seq=27/6912,…
121 814.875165832	10.0.0.2	10.0.0.3	TCP	74 55318 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1…
122 814.875716466	10.0.0.3	10.0.0.2	TCP	54 80 → 55318 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
123 820.005026536	00:00:00_00:00:02	00:00:00_00:00:03	ARP	42 Who has 10.0.0.3? Tell 10.0.0.2
124 820.005389782	00:00:00_00:00:03	00:00:00_00:00:02	ARP	42 Who has 10.0.0.2? Tell 10.0.0.3
125 820.005400329	00:00:00_00:00:02	00:00:00_00:00:03	ARP	42 10.0.0.2 is at 00:00:00:00:00:02
126 820.005552178	00:00:00_00:00:03	00:00:00_00:00:02	ARP	42 10.0.0.3 is at 00:00:00:00:00:03
127 829.985681436	10.0.0.2	10.0.0.3	TCP	74 39036 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1…
128 829.986103384	10.0.0.3	10.0.0.2	TCP	54 22 → 39036 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figura 153. Trafico entre Host 2 y Host 3 al eliminar reglas de Firewall.

Fuente: (El autor)

El resultado al eliminar estar reglas sería el que se muestra en la siguiente figura.

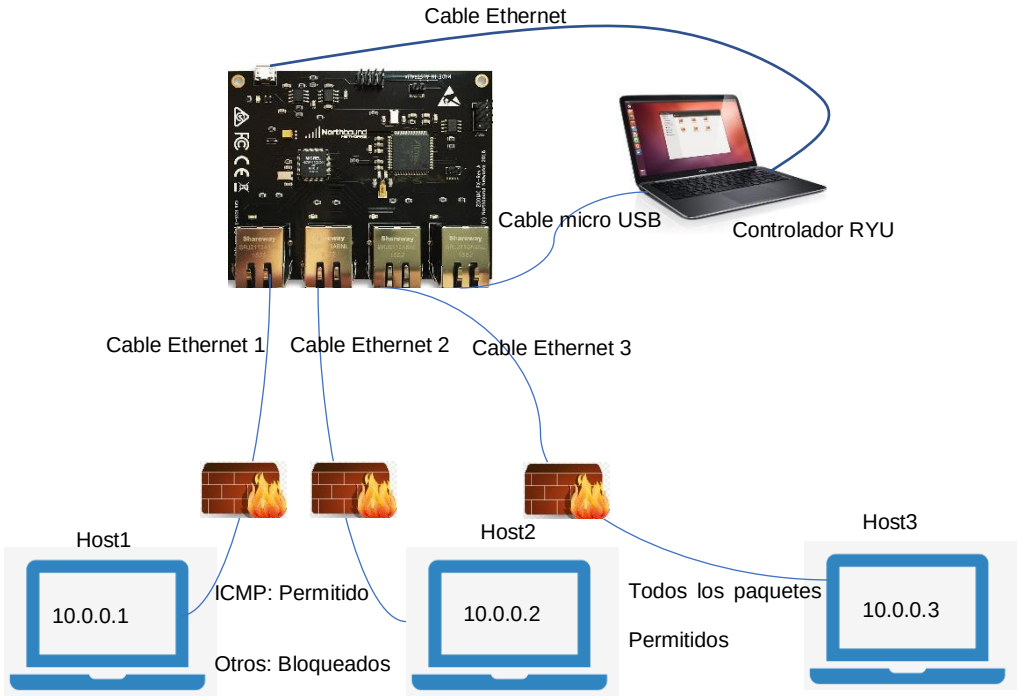


Figura 154. Topología de red al eliminar las reglas de Firewall

Fuente: (El autor)

Conclusiones:

- Se utilizó el Switch Zodiac FX para configurarle la función de Firewall a través de reglas implementadas en una aplicación Python y de esta forma proteger nuestra red a nivel de usuarios finales.
- Se utilizó la herramienta Wireshark para analizar el tráfico de paquetes entre hosts.

Recomendaciones:

- Se recomienda utilizar al menos tres Host para poder observar de mejor forma el comportamiento de las reglas de firewall configuradas.
- Es recomendable ir cambiando las reglas o eliminando las mismas para observar el comportamiento de la red una vez hechos los cambios.

6.2.5 PRACTICA 5:

MONITOR DE TRÁFICO ZODIAC FX

Objetivos:

- Realizar la configuración básica del Zodiac FX para que pueda realizar un monitoreo de tráfico en tiempo real de toda la red conectada al Switch, utilizando Ryu como controlador SDN y 3 Computadoras para el escenario de desarrollo de la práctica.
- Ejecutar la aplicación y ver los resultados que muestra el Controlador al conectar todos los hosts.

Recursos utilizados:

Tabla 19. Recursos Practica 5 Zodiac FX

Recursos de Hardware	**Recursos de Software**
- Computadora Toshiba Satellite Corei7. - Switch Zodiac FX - 3 computadoras de escritorio HP (Host). - 4 cables Ethernet cat. 6	- Sistema Operativo Ubuntu 16.04 - Controlador Ryu - OpenFlow v1.3 - Aplicación Python

Fuente: (El autor)

Procedimiento:

- **Topología:**

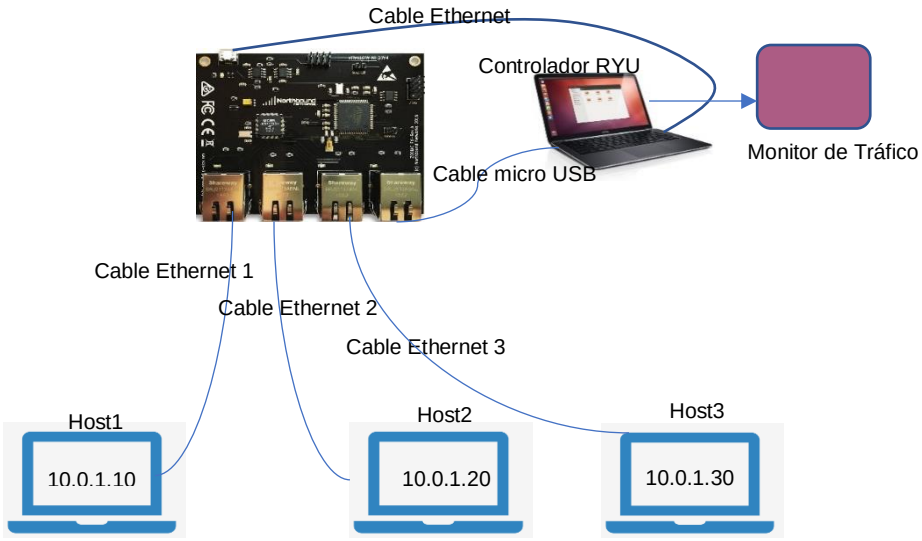


Figura 155. Topología para la Practica 5, Monitor de Tráfico.

Fuente: (El autor)

- **Configuración de host**

  Es necesario configurar una dirección IP para cada host conectado al Zodiac FX, con la finalidad de poder establecer pruebas de eco entre ellos una vez lanzada la aplicación de monitoreo de tráfico.

  El archivo de configuración utilizado para esta práctica se muestra en el **Anexo 7** de este documento.

- **Lanzamiento de la Aplicación**

  Para ejecutar la aplicación se utiliza el mismo arranque proporcionado por Ryu, es decir utilizando ***ryu-manager*** seguido del nombre de la aplicación, esto se muestra en la figura 156.

```
^Cgeovanny@ubuntu-Ryu:~/ryu/ryu/app$ ryu-manager simple_monitor_13.py
loading app simple_monitor_13.py
loading app ryu.controller.ofp_handler
instantiating app simple_monitor_13.py of SimpleMonitor13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 123917683867771 d4:85:64:bf:01:2e d4:85:64:b8:9b:66 3
packet in 123917683867771 d4:85:64:b8:9b:66 d4:85:64:bf:01:2e 1
packet in 123917683867771 d4:85:64:b8:9b:66 d4:85:64:bf:01:2e 1
packet in 123917683867771 d4:85:64:bf:01:2e d4:85:64:b8:9b:66 3
packet in 123917683867771 d4:85:64:b7:fe:00 d4:85:64:b8:9b:66 2
packet in 123917683867771 d4:85:64:b8:9b:66 d4:85:64:b7:fe:00 1
```

Figura 156. Lanzamiento de Aplicación faucet.py.

Fuente: (El autor)

Resultados:

- **Pruebas de Eco entre Host**

  Se realiza un ping entre cada uno de los hosts para poder ver los resultados del intercambio de tráfico en el controlador, esto se muestra en las figuras 157, 158 y 159 respectivamente.

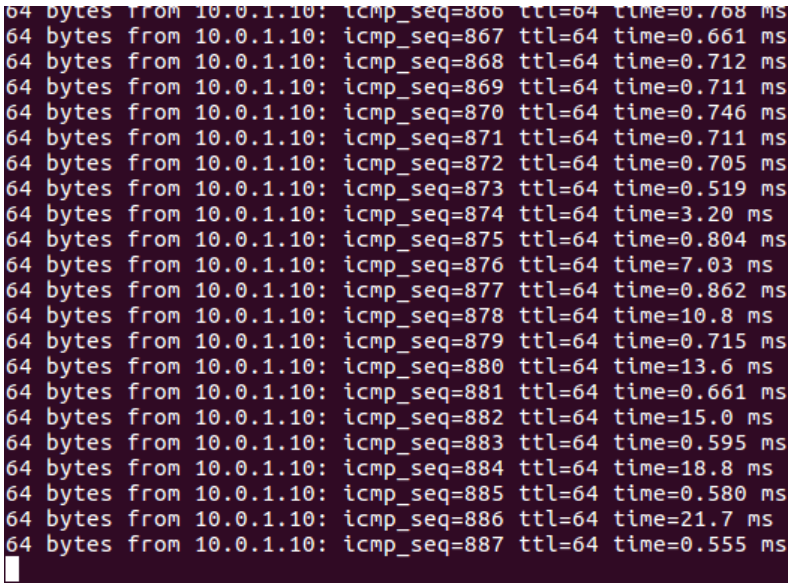


Figura 157. Ping entre host 1 y host 2 Practica 5 Monitoreo de tráfico.

Fuente: (El autor)

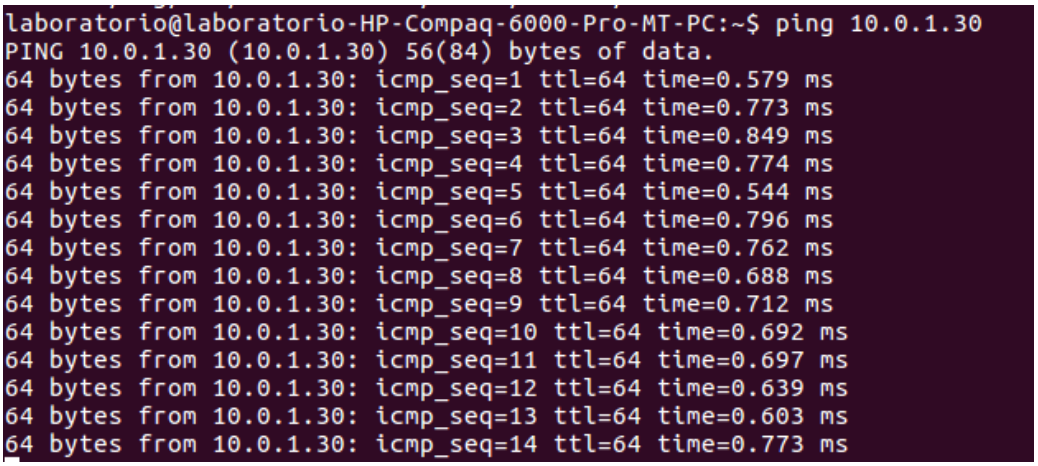


Figura 158. Ping entre host 2 y host 3 Practica 5 Monitoreo de tráfico.

Fuente: (El autor)

```
64 bytes from 10.0.1.20: icmp_seq=1550 ttl=64 time=0.773 ms
64 bytes from 10.0.1.20: icmp_seq=1551 ttl=64 time=0.659 ms
64 bytes from 10.0.1.20: icmp_seq=1552 ttl=64 time=0.830 ms
64 bytes from 10.0.1.20: icmp_seq=1553 ttl=64 time=0.642 ms
64 bytes from 10.0.1.20: icmp_seq=1554 ttl=64 time=0.719 ms
64 bytes from 10.0.1.20: icmp_seq=1555 ttl=64 time=0.487 ms
64 bytes from 10.0.1.20: icmp_seq=1556 ttl=64 time=0.777 ms
64 bytes from 10.0.1.20: icmp_seq=1557 ttl=64 time=0.687 ms
64 bytes from 10.0.1.20: icmp_seq=1558 ttl=64 time=0.563 ms
64 bytes from 10.0.1.20: icmp_seq=1559 ttl=64 time=0.692 ms
64 bytes from 10.0.1.20: icmp_seq=1560 ttl=64 time=0.667 ms
64 bytes from 10.0.1.20: icmp_seq=1561 ttl=64 time=0.823 ms
64 bytes from 10.0.1.20: icmp_seq=1562 ttl=64 time=0.558 ms
64 bytes from 10.0.1.20: icmp_seq=1563 ttl=64 time=0.760 ms
64 bytes from 10.0.1.20: icmp_seq=1564 ttl=64 time=0.748 ms
64 bytes from 10.0.1.20: icmp_seq=1565 ttl=64 time=0.800 ms
64 bytes from 10.0.1.20: icmp_seq=1566 ttl=64 time=0.843 ms
64 bytes from 10.0.1.20: icmp_seq=1567 ttl=64 time=0.801 ms
64 bytes from 10.0.1.20: icmp_seq=1568 ttl=64 time=0.807 ms
64 bytes from 10.0.1.20: icmp_seq=1569 ttl=64 time=0.575 ms
64 bytes from 10.0.1.20: icmp_seq=1570 ttl=64 time=0.567 ms
64 bytes from 10.0.1.20: icmp_seq=1571 ttl=64 time=0.573 ms
64 bytes from 10.0.1.20: icmp_seq=1572 ttl=64 time=0.637 ms
```

Figura 159. Ping entre host 2 y host 1 Practica 5 Monitoreo de tráfico

Fuente: (El autor)

- **Resultados en el Controlador**

El controlador muestra los resultados del intercambio de tráfico entre los hosts que forman parte de la red. esta aplicación actúa como un analizador de paquetes en tiempo real. Los resultados los muestra la figura 160.

```
geovanny@ubuntu-Ryu: ~/ryu/ryu/app
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
000070b3d587407b        1 d4:85:64:bf:01:2e        3      141    13704
000070b3d587407b        2 d4:85:64:b8:9b:66        1      237    23188
000070b3d587407b        2 d4:85:64:bf:01:2e        3       31     3000
000070b3d587407b        3 d4:85:64:b7:fe:00        2       32     3098
000070b3d587407b        3 d4:85:64:b8:9b:66        1      141    13704
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
000070b3d587407b        1     2578   262405        0     2520   260303        0
000070b3d587407b        2     1793   185309        0     1552   148934        0
000070b3d587407b        3     1923   165104        0     1241   130470        0
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
000070b3d587407b        1 d4:85:64:b7:fe:00        2      247    24168
000070b3d587407b        1 d4:85:64:bf:01:2e        3      151    14684
000070b3d587407b        2 d4:85:64:b8:9b:66        1      247    24168
000070b3d587407b        2 d4:85:64:bf:01:2e        3       41     3980
000070b3d587407b        3 d4:85:64:b7:fe:00        2       42     4078
000070b3d587407b        3 d4:85:64:b8:9b:66        1      151    14684
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
000070b3d587407b        1     2598   264037        0     2540   261935        0
000070b3d587407b        2     1813   186941        0     1572   150566        0
000070b3d587407b        3     1943   168368        0     1261   133734        0
```

Figura 160. Resultados de Intercambio de Trafico en el Controlador.

Fuente: (El autor)

Conclusiones:

- Se realizó la configuración básica del Zodiac FX para que pueda realizar un monitoreo de tráfico en tiempo real de toda la red conectada al Switch, utilizando Ryu como controlador SDN y 3 Computadoras para el escenario de desarrollo de la práctica.
- Se ejecutó la aplicación y se reflejó los resultados que muestra el Controlador al conectar todos los hosts.

Recomendaciones:

- Se recomienda conectar los hosts en tiempos separados, con la finalidad de visualizar el intercambio de tráfico de mejor forma.
- Se recomienda conectar y desconectar los hosts conectados al Switch para observar el comportamiento de la red reflejada en el controlador.

7 DISCUSIÓN

Las Redes Definidas por Software (SDN) abren las puertas a una tecnología enfocada a crear código basado en un lenguaje de programación para remplazar o complementar la arquitectura de Redes tradicionales, por esta razón se ha creído conveniente desarrollar un compendio de prácticas utilizando SDN.

Para el desarrollo e implementación de un Prototipo de Redes Definidas por Software se consideró una guía de prácticas divididas en dos secciones. La primera sección que corresponde al uso de SDN utilizando un simulador de redes llamado Mininet y la segunda sección con el uso de un equipo físico como es el Switch Zodiac FX; se utiliza, además un controlador SDN que comparte funciones para el simulador y para el Zodiac FX, este controlador se instaló en una PC con sistema Operativo GNU/LINUX Ubuntu.

Antes del desarrollo del proyecto, fue necesario conocer los conceptos, fundamentos y funcionamiento de SDN con la finalidad de llevarla a cabo en el desarrollo de las prácticas, dando cumplimiento a uno de los objetivos del Proyecto.

Para el desarrollo de la primera sección se plantea cinco practicas relacionadas con SDN, la primera practica corresponde a la instalación y correcto funcionamiento de herramientas necesarias para llevar a cabo el Proyecto en cuestión, dentro de esas herramientas se encuentra el simulador Mininet, el controlador Ryu y herramientas adicionales como Python, Wireshark, Putty, cURL, SAM-BA y otras. Se planteo también practicas SDN como Firewall, Router, IPv6, Video utilizando para su desarrollo Python como lenguaje de programación, Ryu como controlador SDN y OpenFlow como interfaz entre el plano de control y el plano de datos.

La ventaja de utilizar Mininet como herramienta de Simulación es que se puede utilizar un numero bastante amplio de dispositivos de red como Switch, Router y Controladores SDN. Además, ofrece una variedad de topologías que pueden ser utilizadas de acuerdo a la necesidad del usuario. En caso de que no exista una topología acorde a nuestras necesidades, es posible crear nuestra propia topología de red con su propia APP en Python.

Para el dar cumplimiento a la segunda parte del proyecto se ha planteado de igual forma la ejecución de cinco prácticas, dentro de las cuales se empieza por realizar una introducción al funcionamiento de Zodiac FX y las herramientas con las que cuenta este dispositivo SDN. Se agrega además prácticas como Switch, Router, Firewall y Monitor de Tráfico que se llevan a cabo utilizando el dispositivo físico Zodiac FX y las herramientas complementarias descritas con anterioridad.

La ventaja de utilizar Zodiac FX es que siendo un dispositivo SDN, se puede modificar sus características de fabrica tratando de acoplar el dispositivo a la función que se requiera. Además, al tener características físicas bastante simplificadas, es posible crear una red dentro de cualquier escenario.

Uno de los puntos a destacar en cuanto al uso del Zodiac FX es el uso de su interfaz web que es bastante práctica, ya que nos permite seguir de cerca y en tiempo real los cambios que se realiza dentro del dispositivo. Además, ofrece funciones que se pueden hacer de forma manual como configuración de puertos, VLANs, actualización de Firmware, Configuraciones de Red, e incluso ofrece una opción de Reset para volver el dispositivo a su forma original en caso que el usuario lo requiera.

La desventaja de utilizar un dispositivo de estas características es que únicamente está limitado al uso de tres de sus puertos ya que un cuarto puerto cumple la función de conexión al Controlador SDN, conformando lo que es el plano de control, además es posible ejecutar únicamente un proceso a la vez lo que limita sus funciones.

Los resultados obtenidos durante el desarrollo del proyecto han sido satisfactorios, ya que se pudo demostrar los grandes beneficios que trae consigo las SDN, donde destacan características como eficiencia, rendimiento, integridad, bajo costo y que además pueden servir de apoyo para mitigar posibles limitaciones que presentan las redes tradicionales en la actualidad.

Uno de los inconvenientes que se pueden presentar a lo largo del desarrollo de prácticas, es que al ser una tecnología que constantemente está actualizando sus repositorios, es importante tener las versiones más recientes de las herramientas de software utilizadas con la finalidad de evitar inconvenientes al momento de utilizarlas, es decir, podría haber algún tipo de conflicto si alguna versión de software está obsoleta.

Otra de las limitaciones que se puede encontrar es que SDN trabaja con líneas de código generadas en algún lenguaje de programación, dependiendo del controlador que se utilice. Por lo tanto, es importante tener los conocimientos necesarios sobre el lenguaje de programación que se vaya a utilizar, aunque al ser una plataforma de software libre existe una multitud de herramientas e información que ayudan al desarrollador a tener un mejor conocimiento de lo que se plantea hacer.

El desarrollo de este proyecto se realizó con la finalidad de plantar las bases en el tema de Redes Definidas por Software, que puedan ser utilizadas para futuras investigaciones ya sea dentro de la Carrera de Ingeniería en Electrónica y Telecomunicaciones o fuera de ella.

8 CONCLUSIONES

- Se pudo conocer los conceptos, características y funcionamiento de Redes Definidas por Software (SDN), con la finalidad de poder aplicarlos al desarrollo del proyecto en cuestión, permitiendo aclarar temas puntuales de SDN aplicadas al desarrollo del Proyecto en cuestión, basándonos en conceptos aplicados de forma práctica aumentando la fiabilidad de los mismos.

- Se estudio y percibió el funcionamiento del Software de Virtualización Mininet para la simulación y programación de los elementos de Red, con la finalidad de comprender el uso de las diferentes herramientas que incorpora esta herramienta de simulación, ya sea para la creación de topologías de red; así como también para la obtención de un código fuente a través de estas topologías.

- Se desarrollo un compendio de prácticas de Redes Definidas por Software basadas en el simulador Mininet y físicamente a través del Switch SDN Zodiac FX, para futuros desarrollos en la carrera de Ingeniería en Electrónica y Telecomunicaciones, permitiendo establecer una base de estudio para el Desarrollo de futuros trabajos relacionados al uso de Redes Definidas por Software.

- Se diseñó e implementó un prototipo de Redes Definidas por Software (SDN), utilizando Mininet como herramienta de simulación y físicamente con el Switch SDN Zodiac FX, esto permitió experimentar las similitudes y diferencias en cuanto a temas como configuración de dispositivos de red, ejecución de Aplicaciones, configuración y asignación de puertos, establecimiento de reglas de flujo, entre otras.

9 RECOMENDACIONES

- Se recomienda obtener los conocimientos teóricos necesarios antes de empezar con un proyecto de estas características, con la finalidad que no exista confusión al momento de utilizar las diferentes herramientas que ofrece las Redes Definidas por Software.
- Para que no exista algún tipo de conflicto o errores al momento de utilizar las herramientas de Simulación, es necesario consultar los repositorios oficiales donde se alojan los archivos de instalación, con el propósito de obtener información actualizada y confiable.
- Para poder llevar a cabo la creación de aplicaciones en Python, utilizado para el desarrollo de las prácticas propuestas, es necesario obtener un conocimiento previo de este lenguaje de Programación, en caso de no estar familiarizado con el mismo.
- Para el uso del Controlador SDN, es aconsejable tener presente las funciones que nos permite utilizar este, y relacionarlas con las funciones que ofrece el Switch Zodiac FX, de tal forma que se pueda llevar a cabo el cumplimiento de los objetivos planteados durante el desarrollo del proyecto.
- Para poder obtener la información y material necesario para el desarrollo del proyecto se aconseja utilizar documentos científicos, ya que es donde se aloja información relevante a este tema, considerando que es una tecnología aun en desarrollo y no existe información demasiado extensa.
- Se recomienda utilizar herramientas adicionales como analizadores de tráfico, emuladores de terminal y herramientas complementarias a SDN con la finalidad de poder observar el comportamiento de los diferentes escenarios que se va creando durante el desarrollo del proyecto.

10 BIBLIOGRAFÍA

[1] Álvarez, J. (2017). Desarrollo de un Switch híbrido Open Flow/All Path. (Maestría). Madrid, España. Universidad de Alcalá "Escuela Politécnica Superior".

[2] Artimy, M. (26 de 01 de 2019). *Ad Hoc Node*. Obtenido de Building an OpenFlow Switch with Integrated Controller and IDS/IPS: http://adhocnode.com/

[3] Bonilla, J. (2016). DISEÑO E IMPLEMENTACIÓN DE UN FIREWALL L2 UTILIZANDO REDES DEFINIDAS POR SOFTWARE (SDN) (Maestría). Pontificia Universidad Católica del Ecuador, Quito, Ecuador.

[4] Contreras, C. A. (2014). Implementación de un openflow controller para el manejo de openflow switches (Tesis de Pregrado). Bogotá, Colombia. PONTIFICIA UNIVERSIDAD JAVERIANA.

[5] Dignan, L. (17 de Abril de 2012). *Google renueva la red a través de OpenFlow*. Obtenido de ZDNet: https://www.zdnet.com/article/google-revamps-network-via-openflow/

[6] ENGINEERING, C. N. (07 de 03 de 2017). *Introducing Faucet as an OpenFlow Controller*. Obtenido de https://costiser.ro/2017/03/07/sdn-lesson-2-introducing-faucet-as-an-openflow-controller/#.XiyokGhKjIV

[7] España, N. (2016). DISEÑO Y SIMULACIÓN DE UNA RED DEFINIDA POR SOFTWARE (SDN) (Tesis de Pregrado). Universidad Central del Ecuador, Quito, Ecuador.

[8] ESPE. (03 de 03 de 2015). *RYU-Controller Software Defined Networking*. Obtenido de http://190.15.141.68/index.php/espe/ryu-controller

[9] Faucet. (22 de 01 de 220). *Faucet Documentation*. Obtenido de Faucet Developers: https://readthedocs.org/projects/faucet/downloads/pdf/latest/

[10] GitBook. (03 de 05 de 2019). *Zodiac FX + faucet*. Obtenido de https://evacastro.gitbooks.io/sdn/content/zodiac_faucet_rest.html

[11] Gómez, D. F. (2013). Openflow: el protocolo del futuro. *Revista académica e institucional de la UCPR*, (93), 6.

[12] Gonzalez, C. ,. (2018). Evolución y Contribución para el Internet de las Cosaspor las emergentes Redes Definidas por Software. *In Memorias de Congresos UTP*, pp. 28-33.

[13] Guerrero, G. D. (2017). Desarrollo de una plataforma para evaluar calidad de servicios (QoS) en Redes Definidas por Software (SDN) (Tesis de Pregrado). Riobamba, Ecuador. ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO.

[14] Kim, H. &. (2013). Mejora de la gestión de red con redes definidas por software. *IEEE Communications Magazine*, 51 (2), 114-119.

[15] Lamallam, R. (2017). Propuesta de Entorno y Practicas de Laboratorio para Tecnologias SDN. Madrid , España: Universidad Politecnica de Madrid.

[16] Microchip. (20 de 12 de 2019). *SAM-BA In-system Programmer*. Obtenido de https://www.microchip.com/developmenttools/ProductDetails/PartNO/SAM-BA%20In-system%20Programmer

[17] Morillo, D. (2014). Implementación de un Prototipo de una Red Definida por Software (SDN) empleando una solución basada en Software. (Tesis de pregrado). Escuela Politécnica Nacional, Quito, Ecuador.

[18] mylab.net. (04 de 02 de 2018). *RYU SDN CONTROLLER INSTALLATION ON UBUNTU 16.04*. Obtenido de https://rpoernama.wordpress.com/2018/02/04/ryu-sdn-controller-installation-on-ubuntu-16-04/

[19] Navarro, F. W. (2019). Adaptive video transmission over software defined networks. *Visión electrónica*, 13(1).

[20] Nazir, F. H. (2017). Banco de pruebas de red definido por software con ZodiacFX, un conmutador de hardware para OpenFlow. *EAI Endosado Trans. Sistemas de información escalables,*, 4 (14), e5.

[21] Networks, N. (24 de Octubre de 2017). *Kickstarter*. Obtenido de https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch

[22] Networks, N. (05 de 2017). *Zodiac FX USER GUIDE*. Obtenido de https://forums.northboundnetworks.com/downloads/zodiac_fx/guides/ZodiacFX_UserGuide_0517.pdf

[23] Networks, N. (2019). *SDN RESOURCE LIST*. Obtenido de https://northboundnetworks.com/blogs/sdn/sdn-resource-list

[24] Pereira, G. &. (2017). Lineamientos para el Despliegue de Redes SDN/OpenFlow. *Revista Venezolana de Computación* , Vol. 4, No. 2, pp. 21-33.

[25] Planas-Llaudet, A. (s.f.). Configuración de un entorno de emulación que permita el diseño, desarrollo y evaluación de SDN con calidad de servicio (Tesis de Pregrado). *Facultad de Informática de Barcelona, Universidad Politecnica de Catalunya*. Catalunya, España.

[26] Roncero, O. (2016). Software Defined Networking (Maestría). Universidad Politécnica de Cataluña. Barcelona, España.

[27] Rothenberg, C. E. (2010). OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, Campinas,* , 7(1), 65-76.

[28] Ryubook. (06 de 09 de 2019). *Ryubook 1.0 documentation*. Obtenido de https://osrg.github.io/ryu-book/en/html/rest_firewall.html

[29] Sáinz, E. G. (2016). SDN SOBRE REDES IEEE 802.11: SIMULACIÓN MEDIANTE MININET-WIFI (Tesis de Pregrado). *ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN, UNIVERSIDAD DE CANTABRIA*. Cantabria, España.

[30] Sánchez-López, M. (2015). Análisis de redes SDN utilizando Mininet (Tesis de Pregrado). *Universidad de Granada*. Granada, España.

[31] Serrano, D. A. (2016). Redes Definidas por Software (SDN): OpenFlow (Tesis Doctoral). Valencia, España. Universidad Politécnica de Valencia.

[32] Shravanya. (31 de 01 de 2019). *Securing SDN Network using Distributed Controller and Real time Machine Learning*. Obtenido de Implementation of architecture using Zodiac FX switches: https://aristasdn.blogspot.com/2019/01/day-20-implementation-of-architecture.html

[33] Sotelo-Monje, M. A. (2015). ARQUITECTURA PARA LA MEJORA DE LA CALIDAD DE SERVICIOS MULTIMEDIA EN REDES DEFINIDAS POR SOFTWARE (Tesis de Maestría). Madrid, UNIVERSIDAD COMPLUTENSE DE MADRID FACULTAD DE INFORMÁTICA, España.

[34] Tinajero, E. A. (2016). Implementación de un prototipo de switch OpenFlow de bajo costo utilizando una Raspberry Pi (Tesis de Pregrado). Quito, Ecuador. Escuela Politecnica Nacional.

[35] Valencia, C. (2015). DISEÑO Y SIMULACION DE UN PROTOTIPO DE RED DEFINIDA POR SOFTWARE (SDN) USANDO EL PROTOCOLO OPENFLOW. *(Tesis de pregrado)*. Universidad de Guayaquil, Guayaquil, Ecuador.

[36] Yamahata, I. (14 de Septiembre de 2013). *Ryu: SDN framework and Python experience*. Obtenido de https://www.slideshare.net/yamahata/ryu-sdnframeworkupload

[37] Yánez, C. E. (2015). Implementación de un Prototipo de Red Definida por Software para el Hotspot-Espoch Mediante un Controlador Basado en Openflow (Tesis de Pregrado). Riobamba, Ecuador. ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO.

11 ANEXOS

ANEXO 1. Código en Python Práctica Router en Mininet

```
# importación de librerías de Python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import requests
import json

# Inicialización del controlador
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)
# Añadir Switches a la red
  info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)

# Añadir Hosts y direcciones IP a la red
    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host,ip='172.16.20.10/24',  defaultRoute = 'via 172.16.20.1')
    h2 = net.addHost('h2', cls=Host,ip='172.16.10.10/24',  defaultRoute = 'via 172.16.10.1')
    h3 = net.addHost('h3', cls=Host,ip='192.168.30.10/24', defaultRoute = 'via 192.168.30.1')
    h4 = net.addHost('h4', cls=Host,ip='192.168.30.11/24', defaultRoute = 'via 192.168.30.1')

# Añadir enlaces a la red
    info( '*** Add links\n')
    net.addLink(h1, s1)
   net.addLink(h2, s2)
    net.addLink(h3, s3)
    net.addLink(h4, s3)
    net.addLink(s1, s2)
    net.addLink(s2, s3)
    net.addLink(s1, s3)

# Iniciar red
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

# Iniciar Switches
    info( '*** Starting switches\n')
 net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
```

```
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

ANEXO 2. Código en Python Práctica IPv6 en Mininet

```
#!/usr/bin/python
# importación de librerías y códigos base
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
# Inicio del Controlador, protocolo y puerto base
def myNetwork():
   net = Mininet( topo=None,
              build=False,
              ipBase='10.0.0.0/8')

   info( '*** Adding controller\n' )
   c0=net.addController(name='c0',
                 controller=Controller,
                 protocol='tcp',
                 port=6633)
# Añadir Switches a la red
   info( '*** Add switches\n')
   s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
   s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
  s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
# Añadir Hosts y direcciones IPV6 a la red
   info( '*** Add hosts\n')
   h1 = net.addHost('h4', cls=Host, ip=' fc00::1/64', defaultRoute=None)
   h2 = net.addHost('h2', cls=Host, ip=' fc00::2/64', defaultRoute=None)
   h3 = net.addHost('h3', cls=Host, ip=' fc00::3/64', defaultRoute=None)
   h4 = net.addHost('h1', cls=Host, ip=' fc00::4/64', defaultRoute=None)
   h5 = net.addHost('h5', cls=Host, ip=' fc00::5/64"', defaultRoute=None)
# Añadir enlaces entre dispositivos de red
   info( '*** Add links\n')
```

```
    net.addLink(s1, h1)
    net.addLink(h2, s1)
    net.addLink(h3, s2)
    net.addLink(h4, s2)
    net.addLink(h5, s3)
    net.addLink(s1, s2)
    net.addLink(s2, s3)
# Inciar red
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()
# Iniciar Switches
    info( '*** Starting switches\n')
    net.get('s2').start([c0])
    net.get('s1').start([c0])
    net.get('s3').start([c0])
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

ANEXO 3. Código en Python Práctica Video en Mininet

```
#!/usr/bin/python
# Importación de Librerías y Códigos base
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
# Inicio de Controlador, protocolo y puerto Base
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)
# Añadir Switch a la red
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
# Añadir Hosts y direcciones IP a la red
    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
# Añadir enlaces entre dispositivos de red
    info( '*** Add links\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)
```

```
# Iniciar la red
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()
# Iniciar el Switch
    info( '*** Starting switches\n')
    net.get('s1').start([c0])
   info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

ANEXO 4. Código en Python Práctica Switch con Zodiac FX

```
# Importación de Bibliotecas y Códigos Base
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

# Se crea la aplicación Ryu, especifica qué versiones del protocolo OpenFlow con las que la aplicación es compatible e
# inicializa la tabla interna de MAC a puerto.
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

# Controlador de eventos para nuevos conmutadores
# Se ejecuta cada vez que se agrega un conmutador al controlador e instalar una entrada de flujo general.
# permite que el conmutador envíe paquetes al controlador.
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

# Método auxiliar para agregar entradas de flujo
# La mayoría de las entradas de flujo que se agregarán tendrán una estructura similar
# se define un método auxiliar para construir y enviar la entrada de flujo final.
    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
```

```
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

# Aprender la dirección MAC y el puerto asociado
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        self.mac_to_port[dpid][src] = in_port

# Búsqueda de MAC a puerto y destino de paquetes

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # Agregar una entrada de flujo para un destino aprendido
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            # verify if we have a valid buffer_id, if yes avoid to send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)

# Reenviar el paquete enviado al controlador
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)
```

ANEXO 5. Código en Python Práctica Router con Zodiac FX

```
# Importación de Librerías, Bibliotecas y Códigos Base
import logging
import numbers
import socket
import struct
import json
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import dpset
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.exception import OFPUnknownVersion
from ryu.exception import RyuException
from ryu.lib import dpid as dpid_lib
from ryu.lib import hub
from ryu.lib import mac as mac_lib
from ryu.lib import addrconv
from ryu.lib.packet import arp
from ryu.lib.packet import ethernet
from ryu.lib.packet import icmp
from ryu.lib.packet import ipv4
from ryu.lib.packet import packet
from ryu.lib.packet import packet_base
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
from ryu.lib.packet import vlan
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_3

# Parámetros que se pueden definir en una API REST, VLANs, Direcciones IP, Protocolos de Comunicación.
UINT16_MAX = 0xffff
UINT32_MAX = 0xffffffff
UINT64_MAX = 0xffffffffffffffff

ETHERNET = ethernet.ethernet.__name__
VLAN = vlan.vlan.__name__
IPV4 = ipv4.ipv4.__name__
ARP = arp.arp.__name__
ICMP = icmp.icmp.__name__
TCP = tcp.tcp.__name__
UDP = udp.udp.__name__

# Tiempo máximo de espera de transmisión de un paquete de datos
MAX_SUSPENDPACKETS = 50  # Threshold of the packet suspends thread count.

ARP_REPLY_TIMER = 2  # sec
OFP_REPLY_TIMER = 1.0  # sec
CHK_ROUTING_TBL_INTERVAL = 1800  # sec

SWITCHID_PATTERN = dpid_lib.DPID_PATTERN + r'|all'
VLANID_PATTERN = r'[0-9]{1,4}|all'

# Parámetros de configuración de VLAN
VLANID_NONE = 0
VLANID_MIN = 2
VLANID_MAX = 4094

COOKIE_DEFAULT_ID = 0
```

```
COOKIE_SHIFT_VLANID = 32
COOKIE_SHIFT_ROUTEID = 16

# Parámetros de Configuración de Router
DEFAULT_ROUTE = '0.0.0.0/0'
IDLE_TIMEOUT = 1800  # sec
DEFAULT_TTL = 64

# Parámetros de reglas de Configuración REST
REST_COMMAND_RESULT = 'command_result'
REST_RESULT = 'result'
REST_DETAILS = 'details'
REST_OK = 'success'
REST_NG = 'failure'
REST_ALL = 'all'
REST_SWITCHID = 'switch_id'
REST_VLANID = 'vlan_id'
REST_NW = 'internal_network'
REST_ADDRESSID = 'address_id'
REST_ADDRESS = 'address'
REST_ROUTEID = 'route_id'
REST_ROUTE = 'route'
REST_DESTINATION = 'destination'
REST_GATEWAY = 'gateway'

# Asignación de Prioridades
PRIORITY_VLAN_SHIFT = 1000
PRIORITY_NETMASK_SHIFT = 32

PRIORITY_NORMAL = 0
PRIORITY_ARP_HANDLING = 1
PRIORITY_DEFAULT_ROUTING = 1
PRIORITY_MAC_LEARNING = 2
PRIORITY_STATIC_ROUTING = 2
PRIORITY_IMPLICIT_ROUTING = 3
PRIORITY_L2_SWITCHING = 4
PRIORITY_IP_HANDLING = 5

PRIORITY_TYPE_ROUTE = 'priority_route'

# Definición de prioridades a través de Interfaz REST, para tipos de ruta, rutas estáticas, rutas por defecto.
def get_priority(priority_type, vid=0, route=None):
    log_msg = None
    priority = priority_type

    if priority_type == PRIORITY_TYPE_ROUTE:
        assert route is not None
        if route.dst_ip:
            priority_type = PRIORITY_STATIC_ROUTING
            priority = priority_type + route.netmask
            log_msg = 'static routing'
        else:
            priority_type = PRIORITY_DEFAULT_ROUTING
            priority = priority_type
            log_msg = 'default routing'

    if vid or priority_type == PRIORITY_IP_HANDLING:
        priority += PRIORITY_VLAN_SHIFT

    if priority_type > PRIORITY_STATIC_ROUTING:
        priority += PRIORITY_NETMASK_SHIFT

    if log_msg is None:
        return priority
    else:
        return priority, log_msg
```

```
def get_priority_type(priority, vid):
    if vid:
        priority -= PRIORITY_VLAN_SHIFT
    return priority

# Mensaje mostrado en el controlador sobre el estado del Switch Conectado/Desconectado
class NotFoundError(RyuException):
    message = 'Router SW is not connected. : switch_id=%(switch_id)s'


class CommandFailure(RyuException):
    pass

# Mensaje de Informacion que muestra la versión de Openflow
class RestRouterAPI(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION,
                    ofproto_v1_3.OFP_VERSION]

    _CONTEXTS = {'dpset': dpset.DPSet,
                 'wsgi': WSGIApplication}

# Registro e inicialización de la aplicación del controlador
    def __init__(self, *args, **kwargs):
        super(RestRouterAPI, self).__init__(*args, **kwargs)

        # logger configure
        RouterController.set_logger(self.logger)

        wsgi = kwargs['wsgi']
        self.waiters = {}
        self.data = {'waiters': self.waiters}

        mapper = wsgi.mapper
        wsgi.registory['RouterController'] = self.data
        requirements = {'switch_id': SWITCHID_PATTERN,
                        'vlan_id': VLANID_PATTERN}
# Asignacion de reglas de flujo a raves de la interfaz REST
        # For no vlan data
        path = '/router/{switch_id}'
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='get_data',
                       conditions=dict(method=['GET']))
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='set_data',
                       conditions=dict(method=['POST']))
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='delete_data',
                       conditions=dict(method=['DELETE']))
        # For vlan data
        path = '/router/{switch_id}/{vlan_id}'
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='get_vlan_data',
                       conditions=dict(method=['GET']))
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='set_vlan_data',
                       conditions=dict(method=['POST']))
        mapper.connect('router', path, controller=RouterController,
                       requirements=requirements,
                       action='delete_vlan_data',
```

```
                    conditions=dict(method=['DELETE']))
# Controlador de eventos para nuevos conmutadores
    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def datapath_handler(self, ev):
        if ev.enter:
            RouterController.register_router(ev.dp)
        else:
            RouterController.unregister_router(ev.dp)
# Manejador de paquetes y Disección de paquetes
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        RouterController.packet_in_handler(ev.msg)

    def _stats_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if (dp.id not in self.waiters
                or msg.xid not in self.waiters[dp.id]):
            return
        event, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

# Definición de diferentes versiones de OpenFlow 1.0-1.2-1.3
        if ofproto_v1_3.OFP_VERSION == dp.ofproto.OFP_VERSION:
            more = dp.ofproto.OFPMPF_REPLY_MORE
        else:
            more = dp.ofproto.OFPSF_REPLY_MORE
        if msg.flags & more:
            return
        del self.waiters[dp.id][msg.xid]
        event.set()

    # for OpenFlow version1.0
    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_0(self, ev):
        self._stats_reply_handler(ev)

    # for OpenFlow version1.2/1.3
    @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_2(self, ev):
        self._stats_reply_handler(ev)

 # Se carga las tablas de Routing y se muestra el estado de los puertos dependiendo si se encuentra activos o
# inactivos, mostrando mensajes de error en caso de estar inactivos los puertos.

def rest_command(func):
    def _rest_command(*args, **kwargs):
        try:
            msg = func(*args, **kwargs)
            return Response(content_type='application/json',
                            body=json.dumps(msg))

        except SyntaxError as e:
            status = 400
            details = e.msg
        except (ValueError, NameError) as e:
            status = 400
            details = e.message

        except NotFoundError as msg:
            status = 404
            details = str(msg)

        msg = {REST_RESULT: REST_NG,
               REST_DETAILS: details}
        return Response(status=status, body=json.dumps(msg))
```

```
    return _rest_command

# Se define la clase controlador
class RouterController(ControllerBase):

    _ROUTER_LIST = {}
    _LOGGER = None
# Se inicializa la Aplicacion de Controlador
    def __init__(self, req, link, data, **config):
        super(RouterController, self).__init__(req, link, data, **config)
        self.waiters = data['waiters']

# Se define parametros como ID, MAC del Router
    @classmethod
    def set_logger(cls, logger):
        cls._LOGGER = logger
        cls._LOGGER.propagate = False
        hdlr = logging.StreamHandler()
        fmt_str = '[RT][%(levelname)s] switch_id=%(sw_id)s: %(message)s'
        hdlr.setFormatter(logging.Formatter(fmt_str))
        cls._LOGGER.addHandler(hdlr)

# Se realiza un registro del Router en la red
    @classmethod
    def register_router(cls, dp):
        dpid = {'sw_id': dpid_lib.dpid_to_str(dp.id)}
        try:
            router = Router(dp, cls._LOGGER)
        except OFPUnknownVersion as message:
            cls._LOGGER.error(str(message), extra=dpid)
            return
        cls._ROUTER_LIST.setdefault(dp.id, router)
        cls._LOGGER.info('Join as router.', extra=dpid)

    @classmethod
    def unregister_router(cls, dp):
        if dp.id in cls._ROUTER_LIST:
            cls._ROUTER_LIST[dp.id].delete()
            del cls._ROUTER_LIST[dp.id]

            dpid = {'sw_id': dpid_lib.dpid_to_str(dp.id)}
            cls._LOGGER.info('Leave router.', extra=dpid)

# Se define la lista de Routers en la red
    @classmethod
    def packet_in_handler(cls, msg):
        dp_id = msg.datapath.id
        if dp_id in cls._ROUTER_LIST:
            router = cls._ROUTER_LIST[dp_id]
            router.packet_in_handler(msg)

# Se define el formato de asignación de reglas con el Método GET utilizando o no una VLAN

    @rest_command
    def get_data(self, req, switch_id, **_kwargs):
        return self._access_router(switch_id, VLANID_NONE,
                                   'get_data', req)

    # GET /router/{switch_id}/{vlan_id}
    @rest_command
    def get_vlan_data(self, req, switch_id, vlan_id, **_kwargs):
        return self._access_router(switch_id, vlan_id,
                                   'get_data', req)

    # Se define el formato de asignación de reglas con el Método POST utilizando o no una VLAN
    @rest_command
```

```
    def set_data(self, req, switch_id, **_kwargs):
        return self._access_router(switch_id, VLANID_NONE,
                                   'set_data', req)

    # POST /router/{switch_id}/{vlan_id}
    @rest_command
    def set_vlan_data(self, req, switch_id, vlan_id, **_kwargs):
        return self._access_router(switch_id, vlan_id,
                                   'set_data', req)
# Se define el formato de eliminación de reglas con el Método DELETE utilizando o no una VLAN
    @rest_command
    def delete_data(self, req, switch_id, **_kwargs):
        return self._access_router(switch_id, VLANID_NONE,
                                   'delete_data', req)

    # DELETE /router/{switch_id}/{vlan_id}
    @rest_command
    def delete_vlan_data(self, req, switch_id, vlan_id, **_kwargs):
        return self._access_router(switch_id, vlan_id,
                                   'delete_data', req)
# Se define los parámetros que darán acceso al Router
    def _access_router(self, switch_id, vlan_id, func, req):
        rest_message = []
        routers = self._get_router(switch_id)
        try:
            param = req.json if req.body else {}
        except ValueError:
            raise SyntaxError('invalid syntax %s', req.body)
        for router in routers.values():
            function = getattr(router, func)
            data = function(vlan_id, param, self.waiters)
            rest_message.append(data)

        return rest_message

    def _get_router(self, switch_id):
        routers = {}
# Se muestra la lista de Routers dentro de la Red o el no reconocimiento de los mismos a través de un mensaje Error
        if switch_id == REST_ALL:
            routers = self._ROUTER_LIST
        else:
            sw_id = dpid_lib.str_to_dpid(switch_id)
            if sw_id in self._ROUTER_LIST:
                routers = {sw_id: self._ROUTER_LIST[sw_id]}

        if routers:
            return routers
        else:
            raise NotFoundError(switch_id=switch_id)

# Se define la clase Router mostrando parámetros de MAC, ID, O posibles Errores de Configuración
class Router(dict):
    def __init__(self, dp, logger):
        super(Router, self).__init__()
        self.dp = dp
        self.dpid_str = dpid_lib.dpid_to_str(dp.id)
        self.sw_id = {'sw_id': self.dpid_str}
        self.logger = logger

        self.port_data = PortData(dp.ports)

        ofctl = OfCtl.factory(dp, logger)
        cookie = COOKIE_DEFAULT_ID

        # Set SW config: TTL error packet in (for OFPv1.2/1.3)
        ofctl.set_sw_config_for_ttl()
```

```
        # Se define la Prioridad de transmisión de paquetes ARP
        priority = get_priority(PRIORITY_ARP_HANDLING)
        ofctl.set_packetin_flow(cookie, priority, dl_type=ether.ETH_TYPE_ARP)
        self.logger.info('Set ARP handling (packet in) flow [cookie=0x%x]',
                         cookie, extra=self.sw_id)

        # Se define flujos de Switch normal
        priority = get_priority(PRIORITY_NORMAL)
        ofctl.set_normal_flow(cookie, priority)
        self.logger.info('Set L2 switching (normal) flow [cookie=0x%x]',
                         cookie, extra=self.sw_id)

        # Se define flujos de Routeo con VLANs
        vlan_router = VlanRouter(VLANID_NONE, dp, self.port_data, logger)
        self[VLANID_NONE] = vlan_router

        # Se define flujos de Routeo sin utilizer VLANs.
        self.thread = hub.spawn(self._cyclic_update_routing_tbl)
        self.logger.info('Start cyclic routing table update.',
                         extra=self.sw_id)

    def delete(self):
        hub.kill(self.thread)
        self.thread.wait()
        self.logger.info('Stop cyclic routing table update.',
                         extra=self.sw_id)

    def _get_vlan_router(self, vlan_id):
        vlan_routers = []

        if vlan_id == REST_ALL:
            vlan_routers = list(self.values())
        else:
            vlan_id = int(vlan_id)
            if (vlan_id != VLANID_NONE and
                    (vlan_id < VLANID_MIN or VLANID_MAX < vlan_id)):
                msg = 'Invalid {vlan_id} value. Set [%d-%d]'
                raise ValueError(msg % (VLANID_MIN, VLANID_MAX))
            elif vlan_id in self:
                vlan_routers = [self[vlan_id]]

        return vlan_routers
# Se define los parámetros para añadir VLANs
    def _add_vlan_router(self, vlan_id):
        vlan_id = int(vlan_id)
        if vlan_id not in self:
            vlan_router = VlanRouter(vlan_id, self.dp, self.port_data,
                                     self.logger)
            self[vlan_id] = vlan_router
        return self[vlan_id]

    def _del_vlan_router(self, vlan_id, waiters):
        #  Remove unnecessary VlanRouter.
        if vlan_id == VLANID_NONE:
            return
```

ANEXO 6. Código en Python Práctica Firewall con Zodiac FX

```
# Importación de Bibliotecas y Códigos Base
import logging
import json
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.exception import OFPUnknownVersion
from ryu.lib import mac
from ryu.lib import dpid as dpid_lib
from ryu.lib import ofctl_v1_0
from ryu.lib import ofctl_v1_2
from ryu.lib import ofctl_v1_3
from ryu.lib.packet import packet
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_2_parser
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_3_parser

# Definición de parámetros que pueden configurarse dentro de la Interfaz REST utilizada por la Aplicación Firewall
SWITCHID_PATTERN = dpid_lib.DPID_PATTERN + r'|all'
VLANID_PATTERN = r'[0-9]{1,4}|all'

REST_ALL = 'all'
REST_SWITCHID = 'switch_id'
REST_VLANID = 'vlan_id'
REST_RULE_ID = 'rule_id'
REST_STATUS = 'status'
REST_LOG_STATUS = 'log_status'
REST_STATUS_ENABLE = 'enable'
REST_STATUS_DISABLE = 'disable'
REST_COMMAND_RESULT = 'command_result'
REST_ACL = 'access_control_list'
REST_RULES = 'rules'
REST_COOKIE = 'cookie'
REST_PRIORITY = 'priority'
REST_MATCH = 'match'
REST_IN_PORT = 'in_port'
REST_SRC_MAC = 'dl_src'
REST_DST_MAC = 'dl_dst'
REST_DL_TYPE = 'dl_type'
REST_DL_TYPE_ARP = 'ARP'
REST_DL_TYPE_IPV4 = 'IPv4'
REST_DL_TYPE_IPV6 = 'IPv6'
REST_DL_VLAN = 'dl_vlan'
REST_SRC_IP = 'nw_src'
REST_DST_IP = 'nw_dst'
REST_SRC_IPV6 = 'ipv6_src'
REST_DST_IPV6 = 'ipv6_dst'
REST_NW_PROTO = 'nw_proto'
REST_NW_PROTO_TCP = 'TCP'
REST_NW_PROTO_UDP = 'UDP'
REST_NW_PROTO_ICMP = 'ICMP'
REST_NW_PROTO_ICMPV6 = 'ICMPv6'
REST_TP_SRC = 'tp_src'
REST_TP_DST = 'tp_dst'
REST_ACTION = 'actions'
```

```
REST_ACTION_ALLOW = 'ALLOW'
REST_ACTION_DENY = 'DENY'
REST_ACTION_PACKETIN = 'PACKETIN'


STATUS_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX
ARP_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX - 1
LOG_FLOW_PRIORITY = 0
ACL_FLOW_PRIORITY_MIN = LOG_FLOW_PRIORITY + 1
ACL_FLOW_PRIORITY_MAX = ofproto_v1_3_parser.UINT16_MAX - 2

VLANID_NONE = 0
VLANID_MIN = 2
VLANID_MAX = 4094
COOKIE_SHIFT_VLANID = 32

# Registro e inicialización de la aplicación del controlador
class RestFirewallAPI(app_manager.RyuApp):

# Definición de las versiones de OpenFlow
   OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                   ofproto_v1_2.OFP_VERSION,
                   ofproto_v1_3.OFP_VERSION]

   _CONTEXTS = {'dpset': dpset.DPSet,
                'wsgi': WSGIApplication}
# Inicio de procesos
   def __init__(self, *args, **kwargs):
      super(RestFirewallAPI, self).__init__(*args, **kwargs)

# Configuraciones de registro
      FirewallController.set_logger(self.logger)

      self.dpset = kwargs['dpset']
      wsgi = kwargs['wsgi']
      self.waiters = {}
      self.data = {}
      self.data['dpset'] = self.dpset
      self.data['waiters'] = self.waiters

      mapper = wsgi.mapper
      wsgi.registory['FirewallController'] = self.data
      path = '/firewall'
      requirements = {'switchid': SWITCHID_PATTERN,
                      'vlanid': VLANID_PATTERN}

# Asignación de Regla para habilitar o deshabilitar la función Firewall
      uri = path + '/module/status'
      mapper.connect('firewall', uri,
                     controller=FirewallController, action='get_status',
                     conditions=dict(method=['GET']))
# Asignación de reglas para habilitar un flujo de datos
      uri = path + '/module/enable/{switchid}'
      mapper.connect('firewall', uri,
                     controller=FirewallController, action='set_enable',
                     conditions=dict(method=['PUT']),
                     requirements=requirements)
# Asignación de reglas para deshabilitar un flujo de datos
      uri = path + '/module/disable/{switchid}'
      mapper.connect('firewall', uri,
                     controller=FirewallController, action='set_disable',
                     conditions=dict(method=['PUT']),
                     requirements=requirements)
# Asignacion de reglas para registros de Firewall
      uri = path + '/log/status'
      mapper.connect('firewall', uri,
                     controller=FirewallController, action='get_log_status',
```

```
                conditions=dict(method=['GET']))

        uri = path + '/log/enable/{switchid}'
        mapper.connect('firewall', uri,
                controller=FirewallController, action='set_log_enable',
                conditions=dict(method=['PUT']),
                requirements=requirements)

        uri = path + '/log/disable/{switchid}'
        mapper.connect('firewall', uri,
                controller=FirewallController, action='set_log_disable',
                conditions=dict(method=['PUT']),
                requirements=requirements)

# Parámetros de configuración, de reglas utilizando el método GET en la interfaz REST
        uri = path + '/rules/{switchid}'
        mapper.connect('firewall', uri,
                controller=FirewallController, action='get_rules',
                conditions=dict(method=['GET']),
                requirements=requirements)
# Parámetros de configuración, de reglas utilizando el método POST en la interfaz REST
        mapper.connect('firewall', uri,
                controller=FirewallController, action='set_rule',
                conditions=dict(method=['POST']),
                requirements=requirements)
# Parámetros de eliminación de reglas utilizando el método DELETE en la interfaz REST
        mapper.connect('firewall', uri,
                controller=FirewallController, action='delete_rule',
                conditions=dict(method=['DELETE']),
                requirements=requirements)


# Parámetros de configuración, de reglas utilizando el método GET en la interfaz REST para VLAN
        uri += '/{vlanid}'
        mapper.connect('firewall', uri, controller=FirewallController,
                action='get_vlan_rules',
                conditions=dict(method=['GET']),
                requirements=requirements)
# Parámetros de configuración, de reglas utilizando el método POST en la interfaz REST para VLAN

        mapper.connect('firewall', uri, controller=FirewallController,
                action='set_vlan_rule',
                conditions=dict(method=['POST']),
                requirements=requirements)
# Parámetros de eliminación de reglas utilizando el método DELETE en la interfaz REST para VLAN

        mapper.connect('firewall', uri, controller=FirewallController,
                action='delete_vlan_rule',
                conditions=dict(method=['DELETE']),
                requirements=requirements)

    def stats_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

        flags = 0
# Definición de versiones de OpenFlow soportados por el Controlador
    if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION or \
               dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
            flags = dp.ofproto.OFPSF_REPLY_MORE
```

```
        elif dp.ofproto.OFP_VERSION == ofproto_v1_3.OFP_VERSION:
            flags = dp.ofproto.OFPMPF_REPLY_MORE

        if msg.flags & flags:
            return
        del self.waiters[dp.id][msg.xid]
        lock.set()

# Definición de Registros de Control en el Firewall
    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def handler_datapath(self, ev):
        if ev.enter:
            FirewallController.regist_ofs(ev.dp)
        else:
            FirewallController.unregist_ofs(ev.dp)


# Definición de Eventos para la versión 1.0 de Openflow
    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_0(self, ev):
        self.stats_reply_handler(ev)

 # Definicion de Eventos para la versión 1.2 de Openflow
    @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_2(self, ev):
        self.stats_reply_handler(ev)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        FirewallController.packet_in_handler(ev.msg)

# Definicion de la clase Firewall para reconocer la lista de reglas configuradas
class FirewallOfsList(dict):
    def __init__(self):
        super(FirewallOfsList, self).__init__()

    def get_ofs(self, dp_id):
        if len(self) == 0:
            raise ValueError('firewall sw is not connected.')
# Reconocimiento de ID del Dispositivo de red y posibles errores de Registro
        dps = {}
        if dp_id == REST_ALL:
            dps = self
        else:
            try:
                dpid = dpid_lib.str_to_dpid(dp_id)
            except:
                raise ValueError('Invalid switchID.')

            if dpid in self:
                dps = {dpid: self[dpid]}
            else:
                msg = 'firewall sw is not connected. : switchID=%s' % dp_id
                raise ValueError(msg)

        return action
```

ANEXO 7: Código en Python Práctica Monitor de Tráfico con Zodiac FX

```
# Importacion de Bibliotecas
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

# Registro e inicialización de la aplicación del controlador
class SimpleMonitor13(simple_switch_13.SimpleSwitch13):
# se cree un subproceso para emitir periódicamente una solicitud al conmutador OpenFlow para adquirir
# información estadística.
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)
# Se crea un controlador de eventos para adquirir información de tráfico de datos
    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]
# Se define un tiempo promedio para adquirir información de trafico y actualizarla constantemente
    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)
# Se solicita al conmutador información estadística relacionada con la entrada de flujo
    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)
# Se solicita al conmutador información relacionada con los puertos utilizados
    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         '
                         'in-port  eth-dst           '
                         'out-port packets  bytes')
        self.logger.info('---------------- '
                         '-------- ----------------- '
                         '-------- -------- --------')
        for stat in sorted([flow for flow in body if flow.priority == 1],
                           key=lambda flow: (flow.match['in_port'],
                                             flow.match['eth_dst'])):
            self.logger.info('%016x %8x %17s %8x %8d %8d',
                             ev.msg.datapath.id,
                             stat.match['in_port'], stat.match['eth_dst'],
```

```
                    stat.instructions[0].actions[0].port,
                    stat.packet_count, stat.byte_count)
# El monitor mostrara el estado de los puertos y el trafico generado en ellos, además de posibles errores en la red
    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         port     '
                         'rx-pkts  rx-bytes rx-error '
                         'tx-pkts  tx-bytes tx-error')
        self.logger.info('---------------- -------- '
                         '-------- -------- -------- '
                         '-------- -------- --------')
        for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```